

AD-A056 078

BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS
NETWORK OPERATING SYSTEMS.(U)

F/G 9/2

MAY 78 R H THOMAS, R E SCHANTZ, H C FORSDICK
RADC-TR-78-117

F30602-76-C-0425

NL

UNCLASSIFIED

1 OF 4
ADA
056078



AD A 056078

LEVEL 2

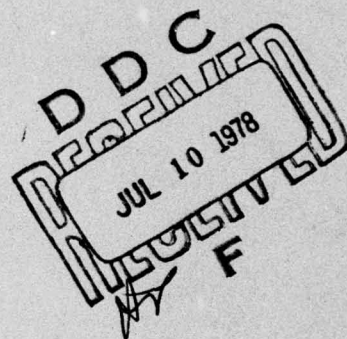
RADC-TR-78-117
Final Technical Report
May 1978



NETWORK OPERATING SYSTEMS

Robert H. Thomas
Richard E. Schantz
Harry C. Forsdick

Bolt Beranek and Newman Incorporated



Approved for public release; distribution unlimited.

AD NO.
DDC FILE COPY

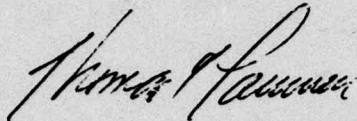
ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

78 07 03 024

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-78-117 has been reviewed and is approved for publication.

APPROVED:



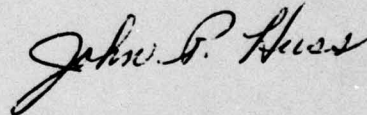
THOMAS F. LAWRENCE
Project Engineer

APPROVED:



WENDALL C. BAUMAN, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER:



JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCP), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| 17 REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|--|--|--|
| 1. REPORT NUMBER RADC-TR-78-117 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) NETWORK OPERATING SYSTEMS. | 5. TYPE OF REPORT & PERIOD COVERED Final Technical Report. Nov 76 - Mar 78 | |
| 7. AUTHOR(s) R. H. Thomas R. E. Schantz H. C. Forsdick | 6. PERFORMING ORG. REPORT NUMBER N/A | 8. CONTRACT OR GRANT NUMBER(s) F30602-76-C-0425 rev |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt, Beranek & Newman, Inc. 50 Moulton Street Cambridge MA 02138 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55971406 | 11. REPORT DATE May 30 1978 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISCP) Griffiss AFB NY 13441 | 12. NUMBER OF PAGES 332 | 13. SECURITY CLASS. (of this report) UNCLASSIFIED |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same | 15. SECURITY CLASS. (of this report) UNCLASSIFIED | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A |
| 16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited. | | |
| 17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same | | |
| 18. SUPPLEMENTARY NOTES RADC Project Engineer: Thomas F. Lawrence (ISCP) | | |
| 19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Networks National Software Works Network Operating Systems Resource Sharing Operating Systems RSEXEC Distributed Computation Multi-Process Systems Personal Computer Systems | | |
| 20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Network operating systems represent a promising approach for realizing the full potential of computer communication networks. A network operating system (NOS) is a collection of software and protocols that allow a set of autonomous computers, which are interconnected by a computer network, to be used together in a convenient and cost effective manner. This report investigates some of the technical problems posed by an NOS by describing and comparing five NOS designs. The system designs considered are: a system for automated terminal access and | | |

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060100

JOB

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

file transfer (ATF); the Resource Sharing Executive (RSEEXEC); the National Software Works (NSW) System; the Efficient Local Access Network (ELAN) System; and the Personal Computer ELAN (PC-ELAN) System. The PC-ELAN System is an NOS designed to support the requirements of collections of personal computers.

| | |
|---------------------------------|---|
| ACCESSION for | |
| NTIS | White Section <input checked="" type="checkbox"/> |
| DDC | B. II Section <input type="checkbox"/> |
| UNANNOUNCED | <input checked="" type="checkbox"/> |
| JUL 1 1981 | |
| DISTRIBUTION/AVAILABILITY CODES | |
| SPECIAL | |
| A | |

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Contents

| | |
|--|-----|
| Executive Summary | 1 |
| 1. Introduction | 8 |
| 2. Previous Work | 30 |
| 2.1. Interprocess Communication | 31 |
| 2.2. Resource Allocation | 32 |
| 2.3. Access Control | 34 |
| 2.4. Fault Tolerance | 35 |
| 2.5. Languages for Distributed Computations | 36 |
| 2.6. Distributed Data Bases: Accessing Mechanisms and Structures | 36 |
| 2.7. Multiple Process Computations | 38 |
| 2.8. Existing Network Operating Systems | 38 |
| 2.9. Assessment | 45 |
| 3. NOS Approaches and Issues | 47 |
| 3.1. Mission Oriented vs General Purpose Systems | 48 |
| 3.2. Implementation Approaches: Base Level vs MetaSystem | 50 |
| 3.3. Implementation Approaches: Single Agent vs Distributed Agent | 55 |
| 3.4. Implementation Approaches: MetaResources | 59 |
| 3.5. General System Characteristics | 61 |
| 3.6. Design Issues: Resource Characteristics | 68 |
| 3.6.1. Files as Network Resources | 69 |
| 3.6.2. Programs as Network Resources | 70 |
| 3.6.3. Devices as Network Resources | 72 |
| 3.6.4. Network Access Control and Resource Allocation | 73 |
| 3.7. Summary | 74 |
| 4. NOS Models - Functional Description | 75 |
| 4.1. A System for Automated TELNET and FTP | 78 |
| 4.1.1. ATF System Objectives | 78 |
| 4.1.2. ATF System Features | 79 |
| 4.1.3. ATF Implementation Approach | 92 |
| 4.1.4. ATF as an NOS | 94 |
| 4.2. Resource Sharing Executive | 97 |
| 4.2.1. RSEEXEC Objectives | 97 |
| 4.2.2. RSEEXEC Features | 100 |
| 4.2.3. RSEEXEC Implementation Approach | 111 |
| 4.2.4. RSEEXEC as an NOS | 123 |

| | | |
|---------|--|-----|
| 4.3. | The National Software Works System | 127 |
| 4.3.1. | Scenario of NSW Use | 128 |
| 4.3.2. | NSW Design Decisions | 130 |
| 4.3.3. | NSW System Structure | 133 |
| 4.3.4. | NSW File System | 140 |
| 4.3.5. | NSW Program Resources | 145 |
| 4.3.6. | System Reliability | 148 |
| 4.3.7. | Access Control and Resource Allocation | 150 |
| 4.3.8. | NSW as an NOS | 152 |
| 4.4. | The ELAN System Design | 159 |
| 4.4.1. | General Assumptions and Characteristics | 160 |
| 4.4.2. | Typical Applications | 164 |
| 4.4.3. | ELAN System Concepts | 166 |
| 4.4.4. | The ELAN File System | 168 |
| 4.4.5. | The ELAN Process Structure | 175 |
| 4.4.6. | Process - File Interaction | 190 |
| 4.4.7. | Use of File and Process Primitives | 203 |
| 4.4.8. | User Interface | 205 |
| 4.4.9. | Access Control | 213 |
| 4.4.10. | Reliability Measures | 217 |
| 4.4.11. | ELAN as an NOS | 227 |
| 4.5. | The PC-ELAN System Design | 230 |
| 4.5.1. | Purpose and General Attributes | 230 |
| 4.5.2. | Typical Applications for Personal Computers | 234 |
| 4.5.3. | Advantages of Personal Computers | 235 |
| 4.5.4. | Existing Single User Machines | 237 |
| 4.5.5. | Personal Computer System Design Constraints | 245 |
| 4.5.6. | Use of Files and Processes on a Personal Computer | 247 |
| 4.5.7. | Integration of ELAN and Personal Computers | 259 |
| 4.5.8. | Communication Requirements of Personal Computers | 265 |
| 4.5.9. | PC-ELAN as an NOS | 269 |
| 5. | Comparison of NOS Models | 271 |
| 5.1. | Role of NOS Administration | 272 |
| 5.2. | NOS Resource Management: Centralized vs Decentralized | 275 |
| 5.3. | Visibility of Distribution | 278 |
| 5.4. | Extendible Program Set | 282 |
| 5.5. | Program Support Services | 283 |
| 5.6. | Interference Between NOS and non-NOS Activity | 285 |
| 5.7. | Implementation Considerations | 288 |

NOS Study

Contents

| | |
|--|-----|
| 6. Implementation Strategies; Requirements for Constituent Host Operating Systems | 290 |
| 7. Conclusions | 299 |
| References | 315 |
| Appendix: State of the Art in Local Area Networks | A-1 |

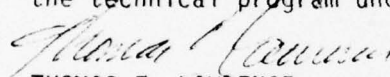
Figures and Tables

| | | |
|------------|--|-----|
| Figure 1. | Typical Network Environment | 9 |
| Figure 2. | Network without an NOS | 14 |
| Figure 3. | NOS Role | 15 |
| Figure 4. | Single Agent Approach to NOS Implementation | 18 |
| Figure 5. | Distributed Agent Approach to NOS Implementation | 19 |
| Figure 6. | RSEEXEC Role | 101 |
| Figure 7. | RSEEXEC File Hierarchy | 103 |
| Figure 8. | Use of Distributed Agent Approach for RSEEXEC | 112 |
| Figure 9. | Use of Monitor Call Intercept Mechanism by RSEEXEC | 119 |
| Figure 10. | NSW System Components | 134 |
| Figure 11. | NSW File Access | 137 |
| Figure 12. | ELAN File Hierarchy | 169 |
| Figure 13. | Two-tiered ELAN File Hierarchy | 172 |
| Figure 14. | Typical ELAN Process Hierarchy | 177 |
| Figure 15. | Distribution of Process in ELAN Process Hierarchy | 178 |
| Figure 16. | User Authentication in ELAN | 210 |
| Figure 17. | PC-ELAN Host Configuration | 249 |
| Table 1. | Single User Computer Systems | 239 |
| Table 2. | Patterns of Communication in PC-ELAN | 267 |

EVALUATION

The Network Operating System (NOS) Study Final Report provides a technical perspective for developments in the NOS area as it applies to distributed heterogeneous computer networks. The report identifies and discusses technical issues, and most importantly, provides recommendations as to which issues require further R&D in order that continued orderly progress may be made.

This effort applies to TPO-5, specifically, Project 2530 "Computer System Reliability and Survivability" (Distributed Data Processing Thrust) and to Project 2531, Task 01 "National Software Works". The information from this study will and has been used in formulating the technical program under Project 2530.


THOMAS F. LAWRENCE
Interactive Processing Section
Computer Technology Branch

Executive Summary

The objective of this study was to investigate the technical problems, alternatives, and approaches for developing a distributed Network Operating System capable of supporting efficient and effective resource sharing in a heterogeneous computer network. It was initiated to extend and refine the research started in a previous "Distributed Computation Study". That study concluded that efficient access and utilization of the resources distributed among a collection of computer systems connected by a communication network requires the development of a Network Operating System (NOS).

For many potential users of a computer network an NOS represents the most effective, and for some perhaps the only, means of using network resources. The fact is that while a network provides the means for resource sharing, a network, even with the standard terminal access and file transfer protocols, does not make such resource sharing easy. In particular: the mechanics of accessing network resources are often difficult to master and tedious to perform; the various hosts and the resources they provide are generally not compatible with one another; information about the resources available, and how to use them, is difficult to obtain; accounting and billing for resource utilization is generally on a per host basis. In short, a network together with its hosts and the resources they provide is not an integrated operating system.

The purpose of an NOS is to act as an agent between users and the resources distributed among the network hosts. It would provide uniform access to the resources, eliminating the need for the user to deal explicitly with network access software, the constituent network hosts, and their operating systems. Although the NOS concept is relatively new, the term NOS has already come to mean a variety of things to different workers, ranging from software that automates the more common network access procedures to total systems that couple the operation of network hosts to implement a fully integrated computer utility.

To help understand and compare the wide range of systems that might be called NOSs, we have identified two fundamental types of NOS services:

- User support services.

To provide these services an NOS accepts commands, such as those required for file maintenance, data access, and program control activities, and interprets the commands in the context of the entire network. The NOS performs network access, host access and data movement procedures required to satisfy the commands.

- Process support services.

To provide these services an NOS supports an execution environment for user processes. Within this environment certain process operations are interpreted in the context of the entire network. These services are provided for standard utility programs such as text editors and language processors as well as private programs written by users. As with the user support services, to provide process support services the NOS must perform network access, host access and data movement operations.

NOS systems can be analyzed in terms of the manner and sophistication with which they provide these services.

NOS implementation approaches are another useful basis for comparing and analyzing different NOS systems. We have identified two fundamentally different approaches:

- Single agent approach.

All NOS functions are implemented by a single software module that occupies a position between the user and the operating systems for the resource bearing hosts.

- Distributed agent approach.

NOS functions are implemented by a collection of software modules distributed among the constituent hosts.

The attraction of the first approach is that it permits relatively inexpensive and quick NOS implementations because it is unnecessary to develop NOS support software for the constituent hosts. The principal disadvantage of the single agent approach is that the types of NOS services it can support are limited. The distributed agent approach can support more sophisticated NOS services but implementations based on it are generally more complex and costly.

A major part of this study involved development of a variety of NOS models. Five models were analyzed and compared; three of them are for NOS system designs developed as part of this study, and two models were developed for existing NOS systems. The systems modeled vary in sophistication, representing different points on the wide spectrum of possible NOS designs. The models are presented in order of increasing sophistication. While this order of presentation is not chronological, each system can be regarded as a logical descendant of the one presented before it

in terms of the concepts it embodies and the NOS services it supports. The system models presented and some of the important NOS concepts they include are:

- Automated TELNET and File Transfer (ATF)

This system design automates the network and host access procedures required to gain terminal access to remote hosts and to transfer files among hosts. It introduces the concept of a user profile for NOS systems.

- Resource Sharing Executive (RSEXEC)

RSEXEC is an existing NOS developed primarily for the TENEX and TOPS-20 hosts on the ARPANET. Important NOS concepts introduced by RSEXEC include a network wide file system, program encapsulation as a means to provide process support services, and device binding.

- The National Software Works (NSW)

NSW is an NOS being developed to permit a heterogeneous collection of ARPANET hosts to be used in an integrated manner to support software production. NOS concepts supported by NSW include centralized system administration and operation, centralized resource management, a general purpose program execution environment, and transparency of host system boundaries.

- Efficient Local Access Network Operating System (ELAN)

ELAN is the design for a general purpose NOS for a collection of heterogeneous hosts. Within ELAN, resource management is distributed in a way that makes high system performance and high system reliability possible.

- Personal Computer-ELAN (PC-ELAN)

This model extends the ELAN design to support large numbers of small personal computers.

System objectives, concepts, functions, and implementation approaches are described for each of these NOS models.

As the result of this study we conclude the following:

- The concepts and system functionality necessary for effective NOS systems are, for the most part, reasonably well understood.
- At present no system exists which provides all or most of the features desirable for an NOS.
- The principal problems preventing development of effective NOS systems lie in the areas of system performance and system reliability.
- In a number of areas mechanisms have been proposed which appear to handle some of the problems raised by an NOS. However, many have not had any real system implementation to test their practicality or to develop the engineering refinements necessary for system integration.
- The use of small personal computers is becoming widespread and will become increasingly more important in the future. These personal computers will be most effective when integrated into and supported by NOS systems.

The first four conclusions, which characterize the state of the art in NOS systems, can be summarized by saying "what to build as an NOS is fairly well understood, but how to build it is not."

In addition, we make the following recommendations:

- A laboratory to experiment with prototype NOS systems should be constructed.

Such a laboratory would make it possible to determine how well various approaches to NOS design problems work together in a realistic environment without the cost commitment required to implement an operational system. NSW, with its underlying communication facility, represents a starting point for such a laboratory. NOS experiments to test and evaluate different NOS approaches could be conducted by modifying or completely replacing various of the NSW system components. To be complete, an NOS laboratory should support experimentation with personal computer systems and various types of computer interconnection strategies, such as local area networks.

- Research into concurrency control mechanisms for distributed data should be pursued.

Successful implementation of most distributed systems hinges on solutions to the problems of data management. Concurrency control is one such problem. A number of approaches to concurrency control for data in a distributed environment have recently been developed. These approaches should be analyzed and classified to identify their similarities and differences, and to identify application classes for which each is best suited.

- The notion of global scheduling for an NOS should be investigated.

It seems clear that for situations in which NOS actions require the participation of several hosts, better NOS performance would result if control could be exerted over the scheduling of the actions by the hosts. It is less clear at present what the nature of this global control should be, how it should be expressed, and how the various host actions should be coordinated.

- Remote record level data access should be investigated.

The ability to access remote data at the record or sub file level (in addition to the file level) is required for efficient NOS operation. Mechanisms that support remote record level data access are not difficult to design or implement. However, strategies for employing record level data access in an effective way in an NOS need to be developed.

- An evaluation of the extent to which automatic data translation can be supported and the role to be taken by the NOS in such translation should be undertaken.

The data translation problem is to maintain the semantic content of data as it is moved from host to host. A number of efforts have demonstrated the feasibility and desirability of establishing networkwide standard file representations and having each host translate into and out of these standards. However, the larger question of what to standardize remains largely unanswered. Furthermore, there is a need to establish a flexible boundary to make provisions for translations best handled by the NOS and those more suitable to the application level software.

- Mechanisms for error recovery and failure tolerance should be investigated with the goal of developing an overall system approach to these problems.

Mechanisms for fault tolerance, where they exist, are for the most part ad hoc and tailored to a specific system. A thorough investigation is needed to determine the general principles underlying the techniques now in use, and to categorize the problems and potential solutions in those areas which have not had adequate attention. Establishing cost parameters for the various mechanisms would also enhance the confidence with which they could be employed in future system designs.

- Research into extensible single host operating systems should continue until the principles and mechanisms are well established and widely available.

One of the strongest lessons learned from various NOS implementation efforts is that hosts that allow users to modify parts of the operating system interface are better able to adapt to support the NOS concept. However, facilities for operating system extensions, where present, are often limited in scope and expensive to use. Research and development in the area of developing modular, extensible computer systems should continue. Additionally, serious consideration should be given to adding some general extensibility features into older host systems whose resources might be particularly useful in a network configuration.

- An effort to develop language support for building and debugging distributed systems should be undertaken.

There is presently little programming language or debugging support for the programming required to develop distributed systems. This is a major factor in the lengthy development periods required for most of these systems. Based on experience gained in building a number of distributed systems, it is now appropriate to consider including some communication, control and debugging abstractions into programming systems. The effort here should attempt to merge network concepts with existing language constructs and develop runtime support flexible enough for the many different types of distributed applications.

1. Introduction

The last decade has seen the rapid evolution of computer communication networks from research curiosities to operational utilities. For example, the ARPANET [67], which was originally developed as a vehicle for research in computer networks and computer resource sharing, currently supports communication between more than 100 computer systems and is used on a daily basis by hundreds of users. Commercial networks, such as TELENET [68] in the U.S. and Datapac [69] in Canada, have very bright futures. The Department of Defense is currently building a large network called AUTODIN II. Based on the packet switching technology pioneered in the ARPANET, AUTODIN II is designed to support the computer communication requirements of the Defense Department in the continental United States well into the 1980's. One attraction of these networks is their ability to provide access to a wide variety of resources distributed among the connected computers.

A typical computer communication network (See Figure 1) includes a communication subsystem to which a collection of computers, called hosts, are connected. This subsystem is generally implemented by communication processors interconnected by communication links, such as coaxial cable, telephone lines, or satellite channels. The communication processors serve two functions. They cooperate to support communication between the hosts, and they provide the interface through which the host

TYPICAL NETWORK ENVIRONMENT

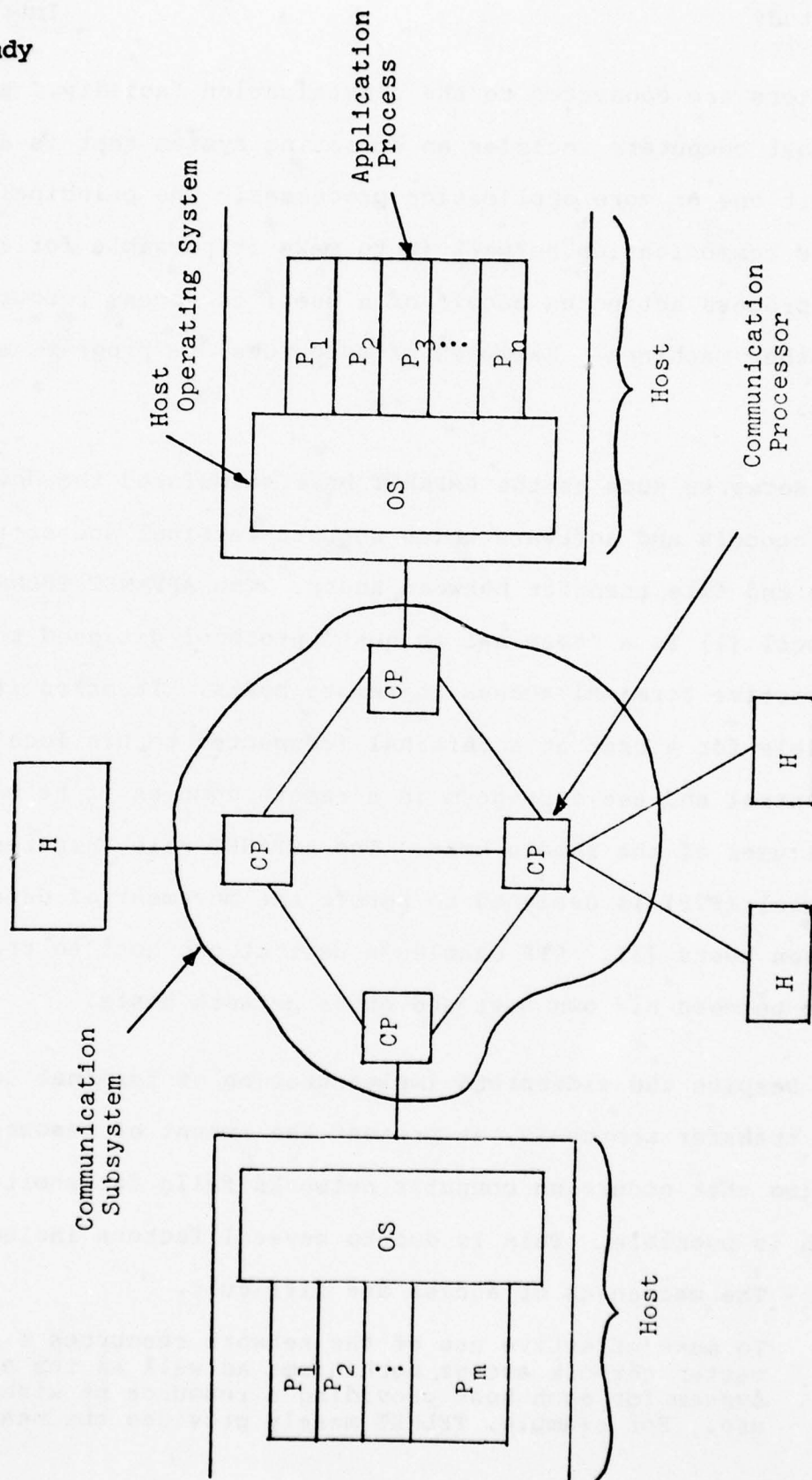


Figure 1.

computers are connected to the communication facility. Each of the host computers includes an operating system that is able to support one or more application processes. The principal purpose of the communication network is to make it possible for a user, or a process acting on behalf of a user, to access resources of the other machines. Examples of resources are programs and data bases.

Networks such as the ARPANET have stimulated the development of protocols and software which support terminal access to remote hosts and file transfer between hosts. The ARPANET TELNET protocol [1] is a "terminal to host" protocol designed to support interactive terminal access to remote hosts. It makes it possible for a user at a terminal (connected to his local host) to control and use a program in a remote host as if he were a local user of the remote host. The ARPANET File Transfer Protocol (FTP) is designed to permit the movement of data files between hosts [3]. FTP enables a user at one host to transmit files between his own host and other network hosts.

Despite the widespread implementation of terminal access and file transfer protocols, at present the amount of resource sharing that occurs on computer networks falls far short of that which is possible. This is due to several factors including:

- The mechanics of access are difficult.

To make effective use of the network resources a user must master network access mechanisms as well as the operating system for each host providing a resource he wishes to use. For example, TELNET merely provides the means for a

user to gain access to a remote host. To make use of the remote host, he must first login to it, supplying a valid name, password and account, and then use the command language conventions of the remote host to initiate a task. Thus, to use the resources of a remote host, the user must master the command language of his own host, the command language of the program on his local host that implements the TELNET protocol, and the command language of the remote host. The situation with respect to file movement is similar. The problem is that the user must explicitly deal with each of the host systems his task involves using their particular and peculiar conventions, and in addition, deal with utility software which supports network access.

- The resources provided by the various hosts are generally not compatible with each other.

A user often encounters great difficulty in attempting to use individual resources for the various hosts together in an integrated fashion. For example, he may want to use the output from a program that runs on one host (e.g., a text editor) as input to a program that runs on another host (e.g., a compiler). Before he can run the second program, the output file produced by the first program must be moved from the first host to the second host by explicit invocation of FTP. Then, if the file structures or data representations supported by the two hosts are different (e.g., sequential versus record structure, 32 bit words versus 36 bit words), the user himself must arrange to convert from one structure or representation to the other.

- Information about the resources available and how to use them is difficult to obtain.

Generally there is no single source that can be consulted for such information. Consequently, users are often unaware of the resources available to them. Even after learning that a particular resource exists and even if the particular resource is well documented, a user must often rely on word of mouth "folk lore" from other users to learn how to use it.

- Accounting and billing for resource utilization is generally on a per host basis.

A user must deal on an individual basis with each of the hosts that manages a resource he plans to use. He must establish an account with each such host before he uses it and arrange to pay for his use of the host according to that host's accounting and billing practices. Then, each

time he uses the host he must supply a valid login name, password and account.

In short, a network, even with its protocol implementations, is not an integrated operating system.

In a previous study entitled "Distributed Computation Study" it was concluded that:

Efficient access and utilization of a collection of computer systems interconnected by a communication network requires development of a Network Operating System (NOS) [24].

The purpose of the study described in this report was to extend and refine the research begun as part of that previous study. In particular, the objectives of this study were to investigate technical problems, alternatives and approaches for developing a distributed, network operating system capable of supporting efficient and effective resource sharing in a heterogeneous computer network. The objectives of this study, as well as the recommendations of the previous study, are discussed in more detail later in this section.

The concept of a "Network Operating System" represents a promising approach for realizing the full potential of computer communication networks. By a network operating system (NOS) we mean a collection of software and supporting communication protocols that allow a set of autonomous computer systems to be used as an integrated facility in a convenient and cost effective manner.

The purpose of an NOS is to act as an agent between users and the resources distributed among the hosts connected to a network. The ideal NOS would provide convenient, uniform and controlled access to the resources; it would act to mediate incompatibilities between the resources so that they can be used together; it would support access to information about the resources and their uses; and it would provide an environment for uniform accounting for resource utilization. To draw an analogy, an NOS would help users and their programs make use of the resources distributed among the network hosts in much the same way a single host operating system provides its users controlled access to local resources.

Figures 2 and 3 illustrate schematically, in a somewhat simplified fashion, the relation of an NOS to a computer network, its users, hosts and resources. Figure 2 depicts the situation a user faces in the absence of an NOS: the network provides access to a wide range of resources resident on a large number of hosts, each of whose operating systems the user must master to use the resources. The role of an NOS is shown in Figure 3. An NOS acts to provide uniform access to the collection of resources, eliminating the need for the user to deal explicitly with the constituent host operating systems and network access software (i.e., TELNET and FTP implementations).

The concept of an NOS is relatively new. However, the term has already come to mean a variety of things to different

NOS Study

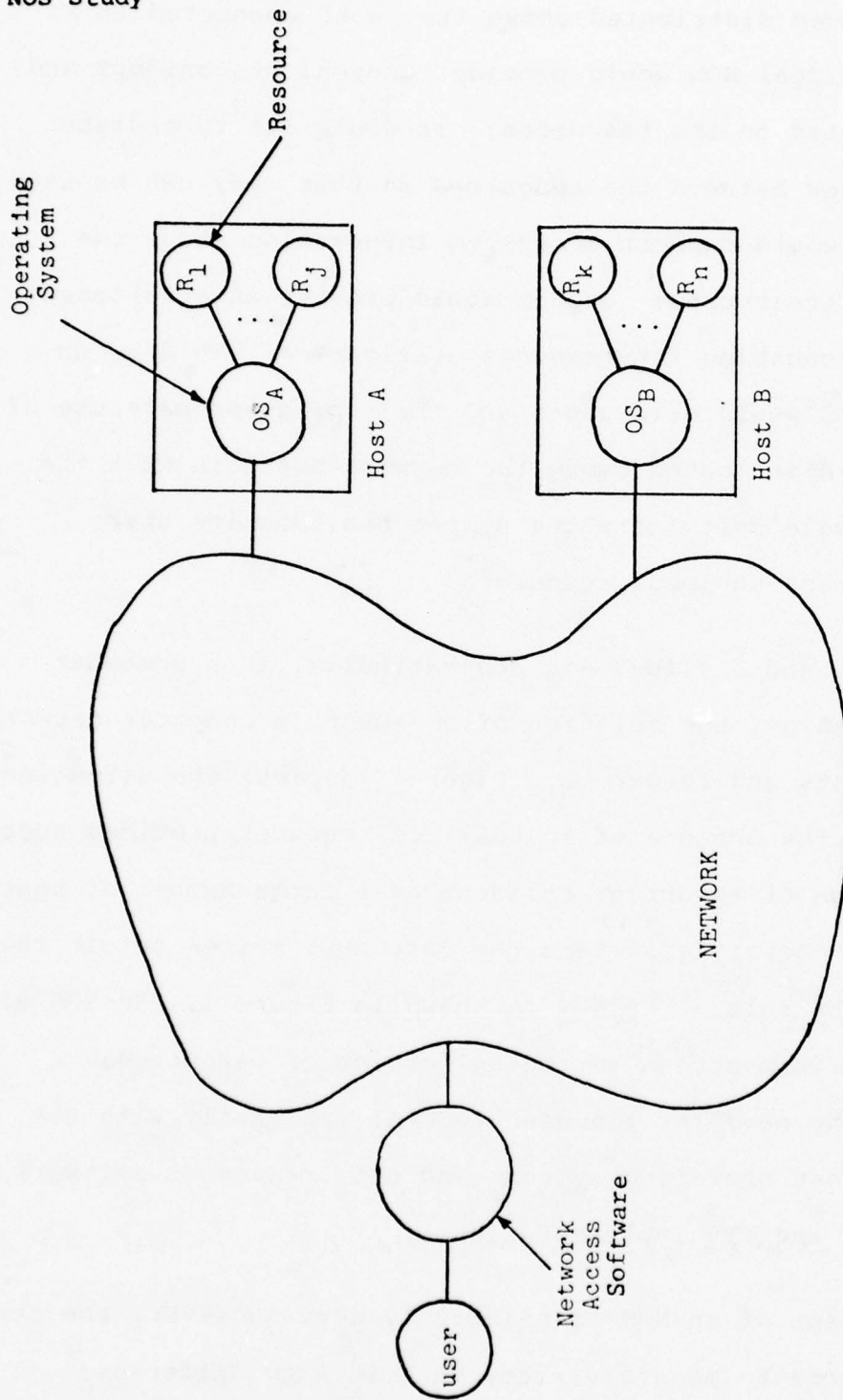


Figure 2.

In the absence of an integrated network operating system, the user must deal with each host separately to access their resources.

NOS Study

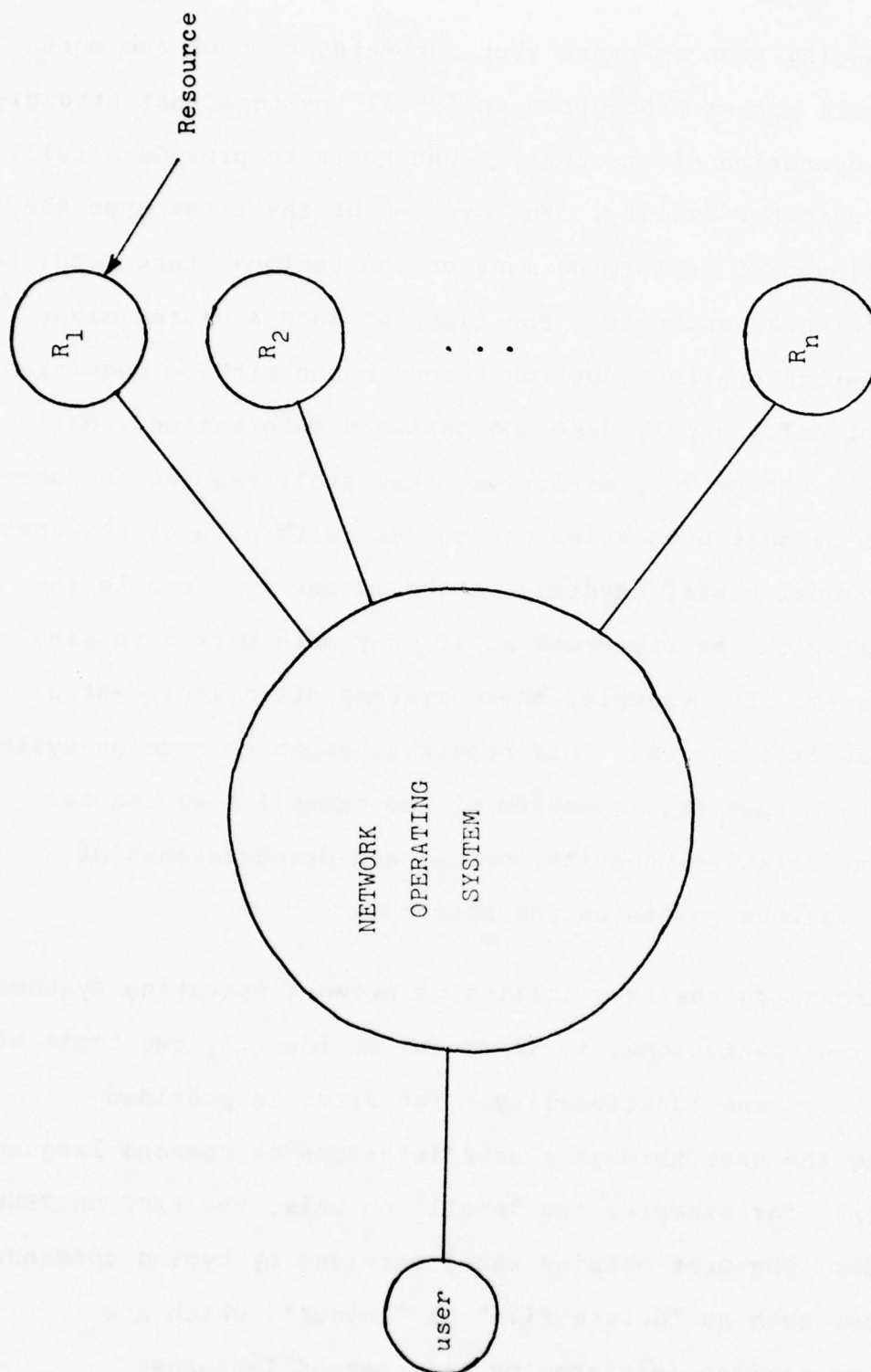


Figure 3.
Conceptual model of NOS role.

workers, ranging from software that automates some of the more common network access procedures to "total" systems that strongly couple the operation of the constituent hosts to provide a fully integrated computer utility. For systems of the first type the user is relieved of performing many of the tedious steps required to access network resources. For example, such a system might enable a user to create "jobs" on remote hosts without requiring them to explicitly supply name and password information. While these systems can be very effective, they still require the user to be aware of host boundaries and to deal with each of the hosts on an individual basis. Systems of the latter type enable the user to deal with the resources as if they were part of a single uniform system. For example, these systems often implement a network-wide file system. This report attempts to examine system concepts which span this spectrum of NOS capabilities and to identify the relative benefits, costs, and disadvantages of systems at various points on the spectrum.

In discussing the capabilities of network operating systems and their implementations, it is useful to identify two types of system services and functionality. The first is provided directly to the user through a user interface or command language interpreter; for example, the "shell" on Unix, the EXEC on TENEX and TOPS-20. The user obtains these services by typing commands or requests, such as "delete file" or "logout", which are satisfied by actions initiated by the command language interpreter. The second type of system service is provided to

programs or processes executing on behalf of the user; for example, text editors, compilers, or programs the user himself has written. The user's program obtains these services by executing operating system calls, such as "open file" or "create process". The operating system software that interprets and satisfies these calls implements the programming interface or program execution environment of the system.

NOS systems of the first type described above typically provide only a user interface. "Total" systems generally support more sophisticated user interfaces. In addition, a total system implements some sort of program execution environment for which operating system calls made by processes are interpreted in the context of the entire network rather than in the limited context of the host where the process happens to be running.

Just as there is a spectrum of NOS capabilities, there is also a variety of approaches to implementing them. Two of the more common approaches are illustrated schematically in Figures 4 and 5. Figure 4 shows what we shall call a "single agent" approach. With this approach all NOS functions are implemented by a single software module that (logically) occupies a position between the user and the operating systems for the resource bearing hosts. This software module implements a user agent that "knows" the conventions of each of the various host operating systems. It acts to transform user requests into the host-specific commands required to satisfy them. Generally, when

NOS Study

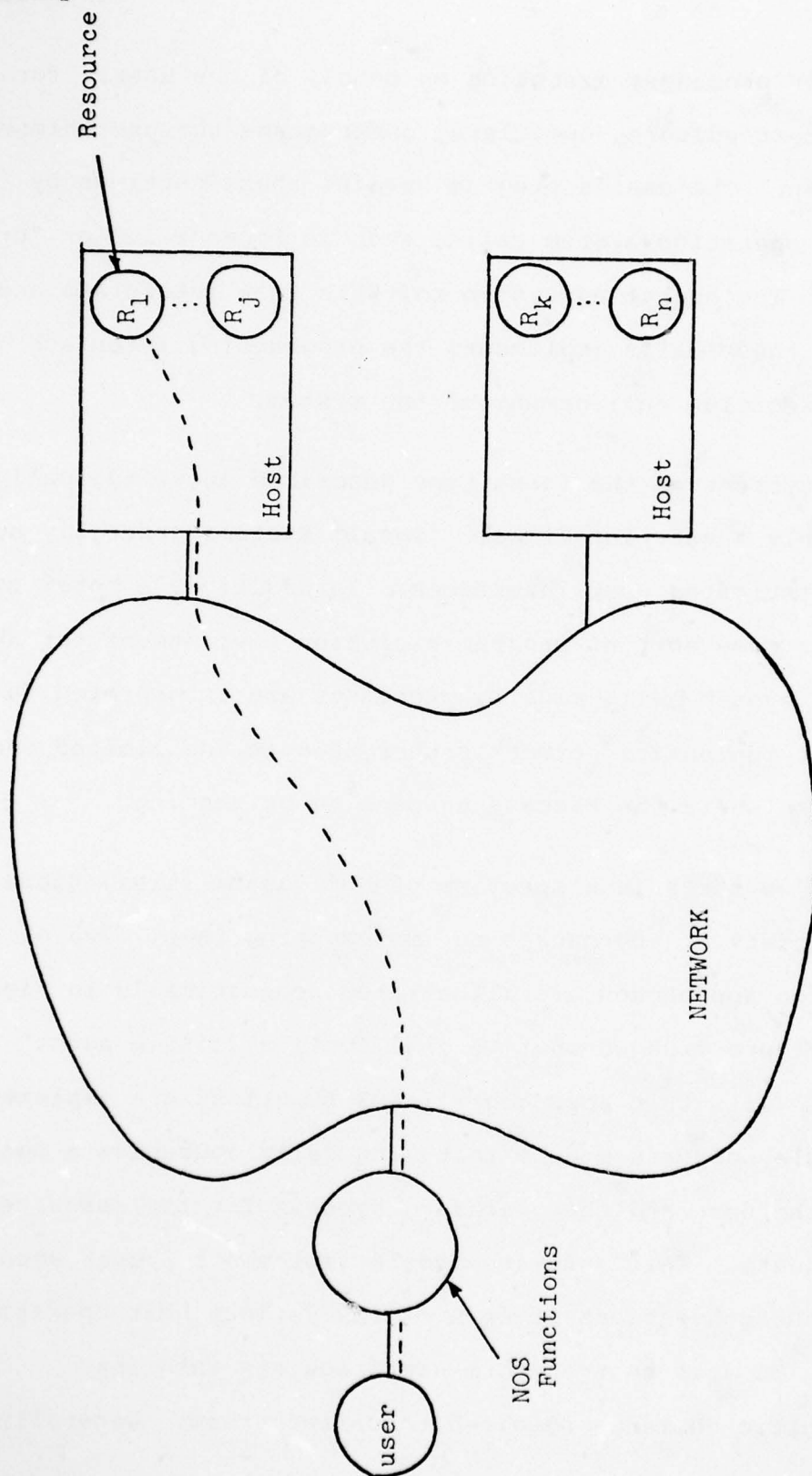


Figure 4.

Single agent approach to NOS implementation.

NOS Study

Resource

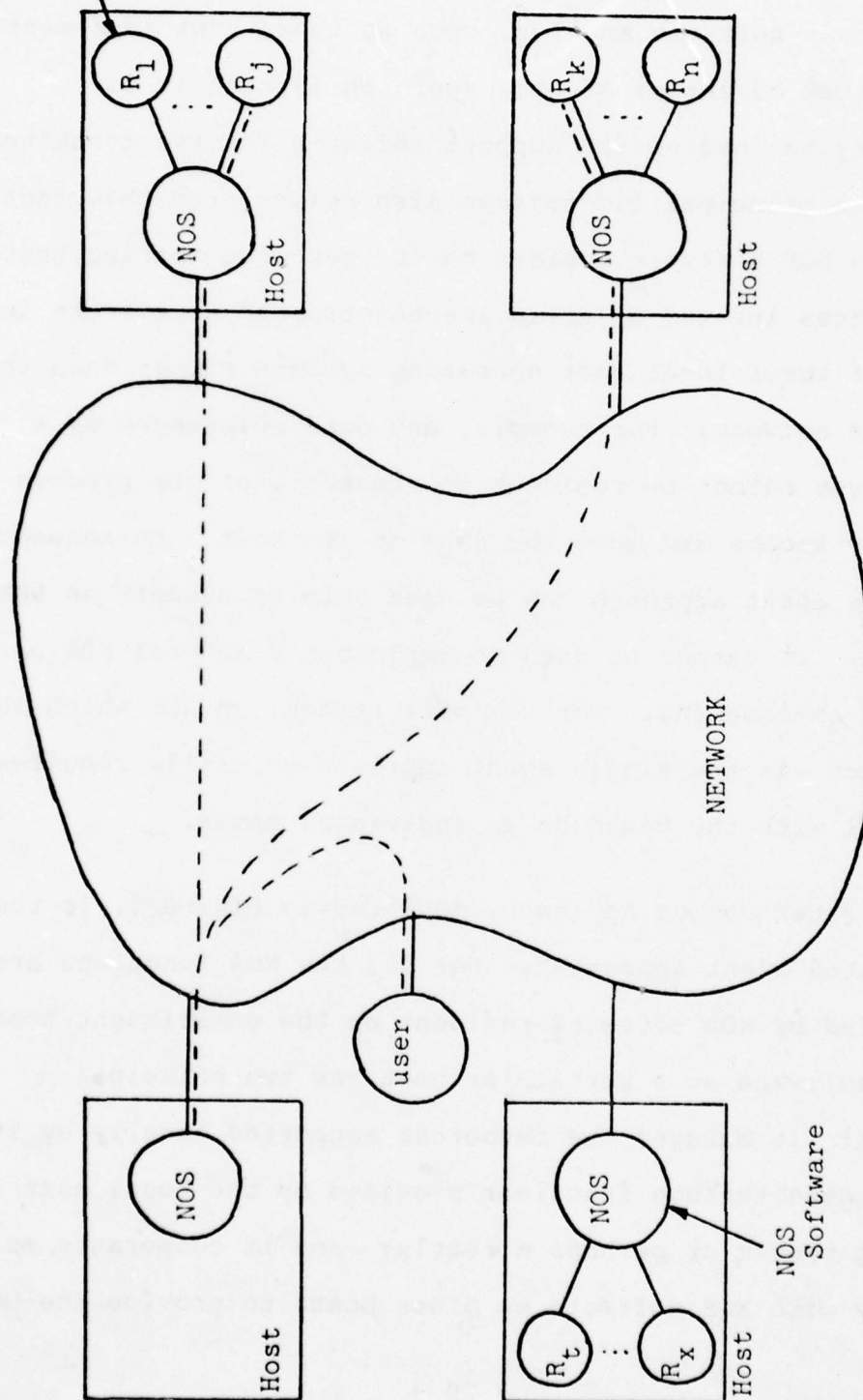


Figure 5.

Distributed agent approach to NOS implementation.

this approach is used, the agent interacts with the constituent hosts through their user or command language interfaces. However, it is also possible for the agent to interact with other standard host software modules, such as those that implement FTP. The principal advantage of this approach is that it is unnecessary to develop NOS support software for the constituent hosts. Its principal limitations also derive from this fact. Because no NOS software resides on the resource bearing hosts, the resources invoked by users are constrained to execute in the context of their local host operating systems rather than that of the entire network. For example, any data referenced by a process must either be resident on its host, or the process must explicitly locate and move the data to its host. Consequently the single agent approach can be used only to support an NOS user interface. It cannot be used to implement a general NOS program execution environment. For the same reason, an NOS which is implemented via the single agent approach generally requires that users deal with the hosts on an individual basis.

The other common approach, depicted in Figure 5, is the a "distributed agent approach". For it, the NOS functions are implemented by NOS software resident on the constituent hosts. The NOS software at a particular host has two principal functions: it manages the resources supported locally by its host, perhaps through functions provided by the local host operating system or perhaps directly; and it cooperates as necessary with NOS software on other hosts to provide the NOS

functions. For this approach all host-specific knowledge is embodied by the NOS module that resides in the host, rather than in the module that implements the user interface. Interactions between the various NOS modules are generally accomplished via standard process-to-process protocols rather than via host-specific user-to-command language interpreter protocols, such as those used in the single agent approach. Because NOS software resides on the resource bearing hosts, it is possible to implement an NOS execution environment for resources with the distributed agent approach. In addition, with this approach it is feasible to implement an NOS which relieves the user from the necessity of dealing with the constituent hosts on an individual basis. The principal disadvantage of the distributed agent approach is that implementations based on it are generally more expensive than those based on the single agent approach.

As discussed above, an important function of an NOS is to act on the user's behalf to provide convenient access to network resources. The extent to which this is accomplished depends upon the sophistication of the NOS. A simple NOS might merely make it easy to create a job and run a program on another host, and, perhaps, to move a data file from one host to another. A more sophisticated one might free the user from learning and remembering the locations for various resources by allowing him to access a resource by specifying its name rather than its host location. Even more sophisticated ones might make the boundaries between host systems and the underlying network transparent to a

user and his programs in the sense that no logical distinction between resources that are local and those that are remote would exist. Such a system would allow use of a collection of resources on a variety of hosts in an integrated fashion without requiring a user to move and translate data files produced by a resource on one host and used by a resource on another.

In addition, depending upon its sophistication, a number of other important benefits can result from an NOS. In principle, an NOS can be designed so that the integration of additional hosts is relatively easy. For a system designed to be extensible, this integration might require installation of NOS software modules into the new hosts and additions to various system configuration and resource management tables. If the additional hosts are the same type as hosts already part of the NOS, no new NOS software would be required. Otherwise, it would be necessary to develop NOS software modules for the new host types to accomplish their integration into the NOS. A number of useful properties derive from this modular extensibility.

Hosts of a given type can be replicated within the NOS to provide alternate sources for the resources they support. If the NOS software is properly structured, this can result in the increased availability of these resources. In addition, the work of providing these resources may be shared among the hosts. A particular resource can be accessible as long as one of the hosts which support it is functioning. Thus, the NOS can be made

resilient to the failure of one or more of the hosts. In a sophisticated NOS it should be possible to switch the users of a host that fails to one of the functioning hosts in a minimally disruptive fashion.

In addition to providing highly reliable access to system resources, an NOS can also support similar access to users' private files. Because an NOS includes many hosts, the probability that a given file will be accessible when referenced can be increased by replicating it at several hosts. An NOS might, for example, support a "multi-image" file feature. A user could declare that certain files were critical and require a particular level of replication. The NOS would then act to ensure that a file so declared was properly replicated and could be accessed as long as at least one of the hosts storing it was properly functioning.

Most of the time, most of the hosts will be functioning properly. At these times it is possible for the NOS to distribute load among them by, for example, selecting more lightly loaded hosts to satisfy new requests for service. The modular extensibility supported by the NOS structure makes it possible to scale the system capacity to match the requirements of the user community. As the load imposed on a given host type grows, new hosts of the type can easily be added to the NOS until the capacity matches or exceeds the load.

There is another possibility for load leveling among hosts of a given type when the host type supports a range of resources. For example, if the host type supports both highly interactive and compute bound resources, the schedulers of some of the hosts can be optimized for interactive service, and the others for compute bound service. The interactive load can then be distributed among those hosts tuned for interactive jobs, and the compute bound load among the others. This functional specialization is made possible by the NOS structure. Here it is exploited for a homogeneous set of hosts to provide better service for different kinds of resource. Below we shall see how it can be used to advantage among a heterogeneous set of hosts.

The set of resources available to users can be enlarged by adding new host types to an NOS. Integration of a host of a new type is more difficult than that of a host of an existing type because NOS software must be developed for the new host type. In addition, some of the NOS software for other host types may need to be augmented. For example, software to convert between file structures or data representations supported by existing hosts and those supported by the new host type which are new to the NOS may be required. However, the required software is well defined and straight forward to implement.

The modular expandability of a well designed NOS makes it possible to integrate a wide range of host types. It also provides the potential for exploiting functional specialization

among hosts of different types to achieve performance improvements. Different host types can be dedicated to support those resources for which they are well matched. To simplify somewhat, N different resource classes could be supported by an NOS that includes N different host types, each type optimized for a different resource class, as an alternative to an NOS that includes N general purpose hosts, each of which can accommodate all the resource classes, but none in an optimal fashion.

At the outset of this study we limited its scope in two ways. First, we decided to consider only general purpose computer communication networks. This eliminates networks such as MININET [25], which are designed for some specialized task or mission. Included are networks such as the ARPANET, TELENET, and AUTODIN II. Because we are most familiar with the ARPANET, we use it whenever necessary as our network model. For purposes of the study we assume that for networks of interest protocols supporting remote terminal access to hosts and file transfer among hosts exist. Again, when it is necessary to be specific we shall use the ARPANET TELNET and FTP as our protocol models. Second, we decided to focus on problems related to general purpose network operating systems rather than those related to special purpose or mission oriented systems.

A reasonable question to ask at this point is: who would use a general purpose NOS and in what ways would it be used? We see several specific situations for which there would be strong

motivation to use an NOS. Users with access to a network but no local host computer of their own, because they have neither financial commitment nor organizational loyalty to their own host, would be motivated to try a variety of network hosts to determine which satisfy their requirements. These users are likely to discover that no single host satisfies all of their needs [76]. An example of such users are those who access the ARPANET through TIPs [72]. Another class of users are those for whom a single network host, perhaps their local host, satisfies the majority of their requirements, but who, from time to time, must use services provided by other hosts. A large corporation or organization, such as one of the Armed Services, with geographically distributed computer installations may come to realize that it is more cost effective to use an NOS to integrate the operation of these installations than it is to provide each installation with all of the resources needed to satisfy its local computing requirements.

It is useful to distinguish three different ways in which computer systems are used: turn-key usage, applications programming, and systems programming. By "turn-key usage" we mean the use of existing services involving little or no programming. Applications programming is the construction of new services, and systems programming is the construction of "system level" mechanisms used by applications programmers to implement new services. Of course, a given user may, at different times, use a system in one, some, or all of these ways. For purposes of

discussing NOS usage, we shall regard systems programming as the task of implementing an NOS and shall consider it no further.

To support effective turn-key usage an NOS should provide convenient simple access to services distributed among various hosts, as well as the ability to manipulate data files on a network wide basis. That is, at a minimum, turn-key users require a sophisticated command language interpreter. To provide adequate support for applications programming an NOS should, in addition, provide an execution environment for the application programs or services. It is true that services that are to execute on a single host don't require more than the execution environment provided by the local host operating system. However, the implementation and use of a service likely to be used in an integrated fashion with other services would be greatly simplified by a network wide program execution environment. For example, use of an application program would be simplified if the user were not required to arrange for the movement of data files accessed by the program to the host on which it executes. This simplification could be achieved if execution of the application program that implements a service resulted in the necessary file movements. Implementing this behavior is significantly easier for the application programmer if the NOS provides an execution environment in which an attempt by a program to reference a file causes the file to be moved and translated as necessary, regardless of the relative locations of the program and file. The simplification here would result

because the program would not itself be required to search through the network for the file and after finding it, to transfer and translate it. The NOS would perform these functions for the application program.

At present no system exists which provides all or even most of the NOS features discussed above. The previous "Distributed Computation Study" [24] concluded that an NOS was required for efficient access and use of the hosts connected to a computer network, and recommended that steps be taken to develop such systems. That study also identified four primary NOS functions: interuser and interhost communication; data migration; job execution and migration; and control. Of these four, that study concluded that the interuser communication function was well understood, but that the others were not and that further research in those areas was required before powerful and effective NOS systems were feasible. Finally, that study recommended the use of network interface machines (NIMs) to facilitate implementation of NOS functions. A NIM would sit between a host and the network and would assume most of the host's responsibility with respect to the NOS.

The present study builds upon the previous "Distributed Computation Study". The purpose of this effort was to investigate the concept of a Network Operating System, as defined above. Its objectives were:

- To evaluate previous work in the NOS area including the "Distributed Computation Study."
- To identify critical NOS functions.
- To develop one or more conceptual models for an NOS.
- To evaluate the models developed in terms of their strengths and weaknesses, and the resources required to implement them.
- To compare the models with each other.
- To identify areas that require further research.

The results of the study are presented in detail in the remainder of this report. Section 2 reviews and evaluates previous work relevant to network operating systems. The major portion of the report explores various NOS concepts in terms of the five system models introduced in the Executive Summary. Section 3 establishes a framework for the models by discussing general NOS issues and a variety of approaches for achieving effective NOS systems. Each of the five models is described in detail in Section 4. Next, Section 5 compares and contrasts the models in a number of areas to illustrate the diversity of approaches to NOS design and the consequences of certain design decisions. Some approaches to NOS implementation are presented in Section 6. Finally, Section 7 presents our conclusions and recommendations.

2. Previous Work

Previous work on network operating systems can be divided into two general categories: general investigations of issues related to distributed systems and design efforts to integrate specific mechanisms into unified network operating systems. Included in the former area are topics associated with conventional, single host operating systems. Many of these standard topics have been reexamined from the point of view of distributed systems:

- Interprocess communication including process synchronization.
- Resource allocation.
- Access control.
- Fault tolerance.
- Languages for expressing (distributed) computations.

Our interest in distributed systems causes us to examine several other topics closely:

- Distributed data bases: accessing mechanisms and structure.
- Concurrency control for replicated or partitioned data bases.
- Data translation between heterogeneous machines.
- Multi-process computations.

In this section, developments in each of the areas mentioned above will be described. The previous work has influenced our approaches in developing the NOS models presented in Section 4.

2.1 Interprocess Communication.

Several efforts have been made to extend the notion of interprocess communication (IPC) into a distributed environment. On the ARPANET, the initial approach to IPC was to provide stream-oriented communication paths, called connections, between remote processors [28]. Soon applications arose where the overhead involved in setting up a connection outweighed the cost of transmitting a short stream of data. Walden [27] proposed a message-oriented IPC mechanism which placed emphasis on developing efficient means of coordinating a message sender with a message receiver. In Farber's Distributed Computer System (DCS) [7] the IPC mechanism took advantage of the DCS ring communication scheme for recognizing the recipient of a message and for message acknowledgement. Part of the interface between the host and the ring contains dynamically alterable tables that record the identifiers of all processes willing to receive messages at that host. An additional idea that came out of DCS is the notion of generic addressing. A message may be directed to any one of a given class of processes.

The NSW Protocol Committee analyzed the patterns of interprocess communication present in typical distributed applications and discovered three types: infrequent short transactions between previously unrelated processors, more frequent, longer transactions between related processes and finally, infrequent, very long transactions. To deal with these

three very different types of communication, a complete package of communication primitives, known as MSG, has been developed [21]. MSG has two forms of process addressing (specific and generic) and three modes of communication (messages, streams and alarms). Together these modes of addressing and data transmission span the communication requirements of many current applications.

The reliability of communication across networks has received a considerable amount of attention. Reacting to many of the initial problems discovered in the ARPANET host-to-host (or NCP protocol), Cerf and Kahn [29] developed the Transmission Control Protocol (TCP). One of the features of TCP is a much more effective error control on communication channels. Reed and Kanodia [75] have worked on the problem of process synchronization where the processes involved are loosely coupled (e.g., resident on different machines) and constrained in the ways they can share data and communicate (as in processes linked together over a low bandwidth, perhaps unreliable communications channel).

2.2 Resource Allocation.

Most research on resource allocation in a distributed environment has concentrated on the optimal placement of data files among a set of hosts connected by a network. Chu [35] studied the problem for a static arrangement of files where the

file accessing patterns were statistically well defined and constant. By employing an integer programming model, Chu's scheme measures the cost of data storage and transmission while still achieving certain minimum file access times and not exceeding the individual storage capacities of the constituent computer systems. Levin and Morgan [38] relaxed the requirement for a well-known single static accessing pattern, substituting for it a multiple segmented measure of file access patterns. The cost of reorganizing the positions of files between segments is introduced to the total cost function. Mahmoud and Riordon [36] make further additions to the model by adding components to the minimization process to make the model conform more realistically to data base management systems.

The allocation of resources other than files (for example, processes) has not received much attention, primarily because tradeoffs in process location are understood even less than file transportation and translation. In the work done on MSG [21], the concept of process allocation (assignment of an existing process or creation of a new process) is coupled with the processing of generically addressed messages. There are attempts within MSG to dynamically route generically addressed messages to the best possible process that can service the message.

2.3 Access Control.

Very little work has been done to extend single host models of access control for distributed systems. Donnelley has proposed a distributed operating system based on capabilities [50]. In this system, capabilities are not only used for access control but also for solving problems associated with naming resources in the system. Using capabilities in this dual fashion was first suggested by Fabry [49] for a single site system.

One of the principal new components present in distributed systems is the communication network to connect the separate sites. One attribute of the connections provided by a network is that they are often not physically controlled by any central administration and thus subject to tapping. Considerable attention has been given to this problem. The problems associated with providing encrypted streams between processes have been studied by Kent [51] and BBN [52]. The results here suffer from the problem that keys must be distributed to each pair of processes that want to communicate. Recently, Hellman and Diffie [53] and Rivest, Shamir, and Adleman [54] developed an encryption scheme that not only solves the key distribution problem, but also allows the sender of a message to add an unforgeable electronic signature.

2.4 Fault Tolerance.

The recent emergence of loosely coupled multiple processor systems (e.g., Pluribus [48] and NSW [18]) has introduced the capability for applications to be composed of truly autonomous parts -- parts that can recover from or tolerate the faults of other parts. The notion of watchdog timers has been integrated into several systems (Pluribus and Number 1 ESS) as a method for error detection. When a timer for a process exceeds its limit, another autonomous process can attempt to perform a corrective action. Cerf and Kahn [29] provided error recovery in the TCP communication protocol in a way that naturally resynchronizes the sender and receiver of messages. This natural error correction mechanism is successful because it is an integral part of the communication protocol, not a feature that was added on to handle errors after the rest of the mechanism was developed. Lampson and Sturgis [47] developed mechanisms that insure consistency in a distributed file system based on the notion of a bimodal write operation: the only possibilities are that a write operation succeeds or doesn't succeed. There are no intermediate states resulting from partially completed writes. The file system is structured so that consistency preserving operations make modifications to a data base by adding updates to an "intentions" list. The contents of the list are applied to the data in one atomic operation which makes use of the bimodal write operation. This way failure in the middle of an update operation leaves the data base in its initial consistent state and allows the update operation to be repeated later.

Research in fault tolerance is still at an early stage. As more experience is gained with multiple autonomous process applications, new methods and approaches should evolve.

2.5 Languages for Expressing Distributed Computations.

Most existing distributed applications were developed without the aid of programming language features designed specifically to support distributed computations. The placement of processing or file storage components has been decided by the applications programmer. Feldman [44] has proposed a system where some of the details of the placement of parts and communication between parts are hidden, although division of the application into parts is still a task that must be performed by the programmer. Languages to support distributed computations are still a fertile research area.

2.6 Distributed Data Bases: Accessing Mechanisms and Structures.

The predominant emphasis in accessing non-local resources has been toward retrieving remotely stored files of data. The model of transporting data to the site where it is to be processed has been an easier one to grasp than a model where computation moves to a data site. The conceptual advantage of distributed data over distributed processes may be due to the relatively modest data complexity of current applications. Applications that employ complex data structures seem to require

more emphasis on computations moving to the site of the data. Deppe and Fry [33] and Peebles and Manning [34] have written overview studies of the current research in distributed data bases. The latter work presents a division of the distributed data management problem into six areas:

- How to provide a unified view of a data base built out of distributed storage elements.
- Where to store data in the system.
- How to locate data.
- How to provide roll-back and recovery.
- How to transfer data to the processing site (including data translation).
- How to provide concurrency control.

The last problem above has received considerable attention recently. When multiple processes attempt to access a data base, their accesses must be coordinated so that they do not interfere with each other. When the data base is replicated for reliability or partitioned because of size constraints, performing this coordination for multiple distributed processes becomes difficult due to the decentralization of the request. There is no single site in the system that can coordinate requests and still have a decentralized system. Johnson and Thomas [40] developed an algorithm for updating records in a distributed data base. This algorithm was only a partial solution to the problem since it could not support general purpose data base locking. Thomas [39] developed a second algorithm that supports general purpose locking while at the same

time demonstrating robust operation in the face of communication or host failures. Similar schemes that emphasize different aspects of the problem have been proposed [42].

2.7 Multiple Process Computations.

Although there has been little work on general purpose methods in this area, the full benefit of distributed systems would be gained by applications that used both distributed processes as well as distributed data bases. Systems have been built that employ both, but general principles for designing and implementing distributed systems have not yet been identified.

Recently, Jenny [74] has described a method for the optimal assignment of subprocesses of an application to processor modules that minimizes interprocessor communications overhead. This work is analogous to the work on determining the optimal assignment of files to processors so as to minimize file movement. Further work needs to be done to develop methods for partitioning computations into separable processes that can proceed autonomously, perhaps on different processors, and in parallel.

2.8 Existing Network Operating Systems.

A number of systems which might be called Network Operating Systems have been developed. Some of these are special purposes systems aimed at a particular task such as data base management or software development while others attempt to serve the needs

of many different types of applications. Quite often new approaches to solving problems or providing usable mechanisms arise only when real systems are actually built. Several significant developments have resulted from the design and implementation of these systems.

The Distributed Computer System (DCS) [7] is a general purpose NOS built on a set of mini-computers. The DCS project was undertaken to explore many of the issues present in distributed systems, especially communication in a distributed system. One of the major contributions of the DCS project was the development of a high-speed communication ring which serves to connect the separate hosts of the system. The fact that messages traveling on the ring pass by every host in the system greatly influences the design of the rest of the system. Strategies that would be inefficient in a packet switched network, such as the use of broadcast protocols, are possible in the ring network when there is no penalty for having a message addressed to all constituent hosts.

A second contribution of DCS is the notion of process addressing. When one process wishes to communicate with another process, the essential attribute is the name of the destination process, not the physical location of the process or the route from the host of the source process to the host of the destination process. In addition, by concentrating addressing on the identity of the destination process, fewer hard to reverse

bindings are established. Thus for interprocess communication to occur over a period of time, the only binding that needs to be maintained is the binding between the process and its identifier. As a result, a process can be moved from one host to another in the middle of an ongoing process-to-process dialog.

There was a conscious effort in DCS to avoid any centralized components, so many traditional operating system functions are implemented as service processes that can have several instances throughout the system. In addition, resources that are logically unique can be implemented by maintaining replicated copies. For example, the file system is distributed throughout the constituent hosts and critical parts stored in duplicate at autonomous hosts to ensure high reliability. A consequence of replicating components is the task of choosing between alternative instances of a service. DCS introduced the idea of issuing requests for bids for services. Presumably, one instance of a service that could respond more quickly or cheaply than another instance would be asked to perform the service. This mechanism could be used to achieve load balancing between the components of the system.

The Distributed Computer Network (DCN) [12] contained many mechanisms (such as process addressing) similar to those of DCS, although the communication medium was not a ring but rather a point-to-point communication discipline. This suggests that the ideas on DCS are not limited to systems that are supported by a

ring communication facility, but rather are useful in many general communication schemes.

The Distributed Processing System (DPS) [11] attempts to aid the implementor of a distributed system by providing a generalized network-wide run-time environment. One purpose behind DPS was to develop general purpose mechanisms motivated by explicit application-specific mechanisms present in early examples of distributed systems. The main emphasis of DPS was on the procedure call as the basis for distributed component communication. When one component wishes to communicate with another, it expresses this communication as a subroutine call, with input arguments, and is prepared to receive the response of the communication as returned values of the subroutine call. To perform the transfer of arguments and results successfully between possibly dissimilar hosts, it was necessary to develop standard data exchange formats for a collection of well-defined data types. In addition, to improve efficiency, standardized descriptions of the disposition of results of a subroutine were developed. Thus when multiple calls are made to subroutines that reside at the same site, the results of one call can be directed as arguments to a second call, thus avoiding the overhead of returning intermediate results.

Two systems, the Network Access Machine (NBS-NAM) [4] developed at the National Bureau of Standards and the Rule-directed Intelligent Terminal Agent (RITA) [6] developed by

the Rand Corporation, attempt to provide uniform access to multiple heterogeneous host computer systems by acting as an intermediary between the user and the systems. Both of these systems make use of unmodified constituent hosts and use string substitution to translate a user's generalized command into host-specific commands. One typical user command is login. A user can issue a generalized command "login Host Username Password", to NAM or RITA which will translate the command into the proper login command for the particular host specified. Both systems provide a capability for analyzing the response to a command so that conditional statements may be executed. These systems are limited solutions to generalized access to network resources because they are able to interact with the constituent host systems at the user interface only.

The Resource Sharing Executive (RSEEXEC) [15] developed at Bolt Beranek and Newman provides network-wide access to resources residing on a set of constituent hosts which primarily run the TENEX [56] operating system. These resources are available both to a human user through a unified user interface as well as to unmodified application programs whose resource references are reinterpreted in the context of the set of resources available on the RSEEXEC constituent hosts. The primary resource type supported by RSEEXEC is files, although limited device and process accessing capabilities are available. RSEEXEC introduced the notion of a unified network-wide file system that a user may treat as an integral entity. The user interface portion of

RSEXEC allows a user to manage a private collection of files that may span many different hosts. The program interface portion allows application programs to access files from the user's private collection, transporting the file if necessary from its storage site to the site of reference. In addition to the distributed file system, RSEXEC provides information to the user about the state of the constituent hosts (e.g., current logged in users and a measure of the system load). Finally, RSEXEC introduced the idea of augmenting the capability of a small host to access the resources of larger hosts. A fuller description of the capabilities and implementation techniques used in RSEXEC is presented in Section 4.2.

The PRIME Computer Company is one of the first manufacturers to provide capabilities for accessing distributed resources within the context of a unified general purpose operating system. Recent releases of the PRIMOS [59] operating system have a facility for storing the contents of a file or a sub-directory on a different host than the one storing the directory entry for the file or sub-directory. A high-speed local ring network connects the hosts together. PRIME is currently designing interprocess communication mechanisms that will allow application processes on different hosts to communicate. The ultimate aim of PRIME is to base its computer systems on inexpensive multiple processing units that are connected by a high-speed network. This architecture facilitates the ability to expand an installation by adding new processing units.

There are a number of examples of special purpose applications that are implemented on a distributed hardware base. One in particular, MININET [25], developed at the University of Waterloo is aimed at a single, large class of applications -- transaction processing. An important example of a transaction processing application is data management. In a data management system a transaction would typically be a request for processing a query or update to the data base. Communication primitives are designed around transactions and are tied to the creation and destruction of transactions -- much as they are in MSG [21]. One characteristic that distinguishes MININET and PRIME from the other network operating systems described above is that they are implemented directly on the hardware rather than on top of an existing operating system environment. For example, in MININET the file system is designed to conform exactly to the requirement of data management applications while in RSEEXEC, the RSEEXEC file system uses the file systems of the constituent hosts to provide storage for RSEEXEC files. Of course the advantage of implementing an NOS on top of an existing system is that advantage can be taken of existing software. A second distinguishing characteristic of MININET is that it is not a general purpose operating system in the sense that new programs cannot be added dynamically to the set of executable programs. The primary implication is that control can be maintained over the types of operations performed in application programs to preserve the efficiency of the entire system.

The National Software Works (NSW) [18] is currently being developed by Massachusetts Computer Associates, SRI International, Bolt Beranek and Newman, UCLA and MIT. Designed as a system to support software production, NSW is built upon multiple heterogeneous hosts. A primary goal of NSW, from the standpoint of distributed systems, is development of a set of techniques to permit the use of independently written application programs (called tools) in an integrated fashion. For one of these techniques, called encapsulation [19] and previously explored in the RSEXEC system, references by the tools to resources are reinterpreted in the NSW accessing environment and mapped into references to resources stored on one of the heterogeneous constituent hosts. As previously mentioned, the MSG interprocess communication mechanism provides a balanced package of communication primitives that allow processes to communicate by one of several methods depending on the amount of information to be transmitted and the expected duration of the communication. Finally, in the file transfer area, there are some rudimentary data transformations that allow data stored in several different representations to be used interchangeably. A detailed description of the goals, functions, and implementation of NSW is contained in Section 4.3.

2.9 Assessment

Despite all of these developments, there are no integrated NOS systems that provide unified, general purpose programming

environments for sets of heterogeneous constituent hosts. Each of the systems described in Section 2.8 addresses some aspects of this goal: RSEXEC provides a distributed file system environment for application programs executing on homogeneous hosts but lacks any facility for dynamic process creation or interprocess communication; DPS is based on a specific single model of the patterns of communication and program modularity; NSW provides a distributed file system and the ability to have non-local processes created dynamically, but this latter mechanism is not intended to be used by the programs developed using NSW and thus the system is not easily extendible.

One of the goals of the study described in this report was to develop models of Network Operating Systems that extend the ideas developed in predecessor systems into an integrated NOS that provides a general purpose interface to the resources of a collection of heterogeneous hosts.

3. NOS Approaches and Issues

Operating systems have traditionally dealt with the problem of transforming a collection of sometimes dissimilar hardware and software resources into an integrated facility that can be used in a convenient and cost effective manner. In this regard an operating system transforms a relatively primitive abstract machine, usually a collection of hardware (1), into another abstract machine better matched to the needs of the user or programmer.

The dramatic advances in computer communication and interconnection technology has lead to the concept of an operating system for a network of computers. Like a single host operating system, a network operating system performs a transformation among abstract machines. However for an NOS there is a wider range of choices concerning the primitive abstract machine upon which to build. Not surprisingly, there are also a number of different views of the type of abstract machine or facility which should emerge from the transformation.

This section begins by outlining a number of different possible approaches to developing the concept of an NOS. Next it discusses some general design issues present in all NOS systems. Finally, by raising a number of questions regarding the nature of

1. The increased use of various levels of microprogramming within computer systems makes this view somewhat simplistic. However, the analogy is useful for our purposes here.

resources present in an NOS, we illustrate some of the areas in which various NOS models may differ.

The intent of this section is to suggest the diversity of NOS systems possible, and to help the reader focus on the issues relevant to classifying them. The five NOS models to be presented in Section 4 exhibit different approaches to many of the issues raised below.

3.1 Mission Oriented vs. General Purpose Systems

We first identify two classes of distributed systems: mission oriented and general purpose systems [24]. Mission oriented systems are designed and implemented to solve a given, well defined problem. They generally are "closed" systems that cannot readily be used outside of the domain of a particular application. Frequently, the problems to be addressed by a mission oriented systems are formulated in terms of specific control and data requirements along with some real time or reliability constraints. The objective of a mission oriented design is the definition of a computer system which satisfies the specified set of application dependent requirements.

General purpose systems have somewhat different goals. The intent here is to build an "open ended" information processing utility which can support the requirements of a wide variety of application domains, some anticipated and others undefined. General purpose systems are most effective when they provide

convenient access to a functionally complete set of operations. These operations can be used to extend or modify the abstract machine provided to the end user (1). A well designed general purpose system can support a variety of such extensions. A key issue in designing a general purpose system is determining a set of operations capable of conveniently supporting many applications. The principal problems here relate to determining what is "convenient" and which requirements of the anticipated applications are sufficiently general to warrant direct support by the operating system. The diversity in approaches to the NOS concept, in part, results from the desire to support general purpose computing and from the open question of the best way to do so.

This study focused on approaches to general purpose NOS systems. Mission oriented systems are difficult to discuss in a meaningful way without in depth familiarity with intended application domains. Therefore, they were considered to be beyond the scope of the study. However, we believe that although the end services they provide are different, there are many design and implementation problems common to general purpose and mission oriented distributed systems.

-
1. In this case the term abstract machine is used to include the environment created by application programs. Thus, for example, an end user who has access to a text editor and formatting program would be considered to have at his disposal primitive operations for document preparation.

3.2 Implementation Approaches: Base Level vs. MetaSystem

There are two fundamentally different approaches to building a general purpose NOS. The two approaches follow from different notions of the resources that constitute the primitive basis for the NOS. In one view, the basic building blocks are interconnected bare computer systems, generally hardware with little or no supporting software, whose configuration and method of interconnection can be specified as part of the design effort. Each system, taken separately, is what is generally the starting point for the development of a conventional, single host operating system. The network operating system would be a collection of system components specifically designed to function effectively and efficiently together. Taken individually, the hardware and software for a constituent machine need not constitute what one normally considers a complete computer system. Taken as a whole, the system components implement a complete, integrated network operating system. We shall call this the base level approach to NOS implementation.

The other approach to NOS implementation takes a different view of the starting point. Rather than creating a system from base level hardware components, it utilizes existing software bases in combination to develop the NOS. The building blocks for such a system are the operating systems of the constituent hosts, or some minor variant of them. We shall call this the meta-system approach to NOS implementation. For this approach

the NOS acts principally to coordinate the activity of the existing host operating systems to provide an integrated computational facility.

Since the base level approach is the least constrained technically, it can be expected to result in the most effective and efficient NOS implementations. A major disadvantage of this approach is that the development costs are relatively high. These high costs can be reduced somewhat by limiting the hardware base to a set of homogeneous nodes. However, doing so limits the variety of different resources available to those that can be supported on the strictly homogeneous hardware base. With a heterogeneous hardware base there is the potential for a greater variety of resources. However, the implementation costs for the NOS will be greater since system software must be developed for each of the different hardware configurations represented in the hardware base.

A second, perhaps more important, disadvantage of the base level approach is that the resulting system is likely to make obsolete the large inventory of application level software which has already been developed to run under the "standard" single host operating systems for many of the constituent hardware configurations. In general, since the single host systems have been discarded, there is no means to use this software without the additional cost of reprogramming it or developing emulators for it. (It is true, of course, that the NOS system software

could provide all the standard single host system functions. However, this seems to be a very expensive way to achieve one of the benefits of a meta-system implementation.)

Despite these disadvantages, there are circumstances where the base level implementation approach makes sense. Special purpose applications are examples, where comparable non-network based facilities do not already exist and where there is little or no investment in existing software. The MININET system, developed at the University of Waterloo, is an example of an NOS built from a homogeneous hardware base to support transaction processing for data base applications. The base level approach is also appropriate for systems whose objective is to support experimentation with interconnection architectures. In this case, development costs and the preservation of previously developed software are generally not issues. The experimentation may range from extending existing models of computation in order to take advantage of distributed environments (e.g., process migration, redundancy of components, host specialization), to developing and evaluating new computational models for distributed environments. Examples of systems that support the former are the DCS project at U.C. Irvine [7] and the DCN project at the University of Maryland [12]. Examples of new computational models that may be of utility in a distributed environment include the data flow model developed by Dennis [45], the ACTOR model [46], and the model being developed at the University of Rochester by Feldman and Rovner [44].

There are two principal advantages to the meta-system approach. First, it does not represent a radical departure from the abstract machines already supported for the constituent systems. An NOS implemented with the meta-system approach generally extends the domain of previously defined resources and structures to span host boundaries. For such a system, existing application software can coexist with new applications written for the distributed environment. Thus, it is possible to preserve existing software and data bases. Perhaps equally important, because it merely extends already familiar system concepts rather than completely redefining them, it is relatively easy for users to move from a single host system to the NOS.

The second advantage of the meta-system approach is that the implementation effort required for it is generally much less than that for the base level approach (for end systems of comparable size and functionality). Much of the existing operating system software can be retained to support the local abstract machine. Modifications and additions to augment the abstraction beyond the host boundaries can also be more easily accomplished in increasingly sophisticated increments, since the basic supporting environment already exists. By and large, most of the machinery for supporting the local abstract machine remains unmodified. With the meta-system approach it is possible to achieve a relatively rapid integration of utilities which have already established their values individually. The result of this integration is an expanded pool of resources beyond those available locally in any of the constituent systems.

A meta-system implementation of an NOS relies on the underlying operating systems to perform many conventional operating system functions, such as processor and memory management. The focus of a meta-system implementation is to expand system functionality to include new features useful in a network environment, such as distributed file systems and interhost interprocess communication.

The limitations of the meta-system approach result from its use of existing operating systems as the basis for NOS implementation. A major problem derives from the heterogeneous nature of the system resources. Incompatibilities among the constituent hosts are the chief concern. Incompatibility is not a problem for a homogeneous meta-system NOS because it is possible to adopt the existing abstract machine as the NOS standard. However, as with a homogeneous base level NOS, for a homogeneous meta-system NOS the set of resources that can be made available is limited. In general, the more integrated the desired NOS facility, the greater the impact incompatibilities among the different systems have on the implementation.

In a base level implementation compatibility among higher level resources, such as processes, files, messages and so forth, can be achieved by software convention and fiat despite the presence of a heterogeneous hardware base. This is generally difficult to do with a meta-system implementation, particularly if a system goal is to preserve existing application software. A

major task in designing a heterogeneous meta-system NOS is the definition of appropriate systemwide resources and the corresponding operations for manipulating them in a way that allows a wide class of host systems to easily incorporate these resources and operations into their existing abstract machines.

3.3 Implementation Approaches: Single Agent vs. Distributed Agent

One approach to building an NOS is to try to limit all NOS development activity to a specific host. This host would be responsible for providing users convenient access to the resources of the other network hosts. This is the single agent approach and was introduced in Section 1. The designated host would act as the user's agent, and is frequently called a Network Access Machine (NAM) because it is his means of access to the network resources.

The NAM approach is attractive when it is difficult or impossible to modify software on the network resource bearing hosts. Since all implementation must be within the NAM, the NAM interface to the network hosts is usually through their user oriented command interfaces. The major function of a NAM NOS is to assist a user in the routine tasks of host selection, establishment of user-to-host communication, and host authentication that must be performed before accessing a resource on the resource bearing host. A sophisticated NAM would accept as commands requests for resources and transform them into the

individual host commands needed to access the desired resources.

The single agent or NAM approach generally needs some sort of user profile for storing frequently used parameters, such as those required to gain access to remote resources. These parameters typically include account names, passwords, and the names of frequently used programs and files. Other important parameters for a NAM oriented NOS might be machine dependent rules for governing the user interaction scenarios needed to imitate actual user input.

Because of the human orientation of most host command languages and the differences between the languages for different hosts, NAM implementations are often complex. A NAM stages requests for remote systems based on internally held parameters. Since no protocol standardization beyond TELNET and FTP is attempted (or feasible, given the constraint that no NOS software can exist on the resource bearing hosts), the NAM must deal with each network host in its own dialect. Consequently, integrating a new host type into a NAM NOS may be a non-trivial task. Furthermore, an implementation which is based on a non-standardized user level protocol cannot adequately provide uniform or sophisticated handling of the wide variety of error situations that can arise.

The major asset of the single agent or NAM NOS approach is that it can provide some attractive user level network access capabilities for a relatively low implementation cost and with no

loss of the autonomy of the resource bearing hosts. Indeed, the resource bearing hosts need not even be aware that they are part of an NOS.

The single agent or NAM approach is generally adequate to handle the bookkeeping associated with gaining access to a network resource. The fact that the implementation must be totally within the accessing machine which interacts individually and exclusively with other network sites is architecturally very limited. In order to provide a programming interface to the set of network resources, or to support the kinds of user level resource sharing commonly provided on current single host computer systems (e.g., controlled file sharing), a system architecture with NOS agents on each of the resource bearing hosts is required.

There are two fundamental changes that result from a switch to the distributed agent approach. The first is the possibility for a program-to-program interface between the distributed NOS agents on the constituent systems. This interface can be designed to match the requirements of the NOS. The second change is the possibility of supporting richer patterns of host interactions. No longer is it necessary to have a single designated host interact individually with the other hosts. Resource bearing hosts can now interact with each other to support the integrated NOS concept. For a single agent approach there is a single point of control where it is convenient (and

necessary) to perform the ad hoc host specific translations required by the absence of standardized protocols. The fact that the distributed agent approach no longer constrains the implementation to a single control point tends to diminish the desirability of system by system translation techniques since there are so many more translations potentially required. What is needed is a uniform, standard networkwide protocol to support resource sharing. The same arguments which motivated the standardization of the lower level communication functions supported by the host-to-host and TELNET protocols in the ARPANET also motivate higher level resource sharing protocols for tasks such as running a program and accessing remote files.

By taking the step of providing long lived server processes (the distributed agents) on the constituent NOS hosts, we introduce the opportunity of establishing protocols to coordinate their actions. A key issue for such a protocol is the definition of the objects or resources it is to manipulate. One approach is to define the protocol around those concepts and entities which are common to most computer systems, such as files, processes and job execution. A further refinement would be to recognize at the protocol level structure within files. Regardless of the specific details, the characteristics of the protocols determine the extent to which the resources can be manipulated across host boundaries and, therefore, the extent to which use of the resources can be integrated within the NOS.

3.4 Implementation Approaches: MetaResources

A homogeneous environment is far simpler from the point of view of selecting an appropriate set of resources and attributes to be defined within the NOS protocols. For a homogeneous system the existing abstract machine representations for the individual resources can be supported. An implementation technique well suited for a homogeneous NOS is to directly use the local operating system software to support remote access to local resources. This maximizes the use of existing support software and minimizes NOS development effort.

A heterogeneous configuration presents the problem of determining common attributes for different implementations of similar objects to permit interchangeable use of these seemingly similar, but different, resources. An approach to this problem is to have the NOS support special classes of resources called meta-resources, which are built from the resources supported by the constituent hosts. All instances of a particular meta-resource, such as a file in an NOS file system, would have a defined set of properties in common, and would be interchangeable to the level addressed by the networkwide specification of the particular meta-resource class. If a meta-resource approach is adopted to lessen the impact of incompatibilities among the constituent hosts, other issues arise. The meta-resources will typically have different properties than the "local" versions of the same resource. Should the NOS provide access to both types

of resource separately? The desire to utilize existing software written for the local host system and the "local" versions of resources, and the desire for a uniform set of system resources sometimes conflict in these areas. The technique of program encapsulation (to be discussed in Section 4), which involves automatic translation from one set of resource conventions to another, has proven to be a valuable tool for achieving uniformity while supporting existing programs.

Introducing NOS meta-resources may ultimately have interesting side effects. As meta-resources are defined which differ more and more from the existing local resources one might expect new software to be developed to directly manipulate the meta-resources. If this becomes a trend it will tend to diminish the need to preserve old non-network oriented software. Thus, we may rely less on the actual properties of the underlying host systems and more on the properties provided by the NOS itself. As more of the application programs are supported directly by the NOS rather than directly by the host systems, a more integrated NOS utility evolves. The extreme for such an evolutionary development would be a system which ultimately is unconstrained by the existing host operating systems. The result would be an NOS quite similar to one developed using a base level implementation. If this point is ever reached, in a sense we will have come full circle.

3.5 General System Characteristics

There are a number of design issues that need to be addressed regardless of the approach chosen for NOS implementation. Some of the issues represent what might be considered to be goals for many types of NOSs. Others represent points in the design process where decisions must be made to better match the NOS to a particular environment or set of requirements. We briefly introduce the issues and explain some of the consequences of various approaches to them. They will also be discussed in Section 4 as they specifically relate to each NOS model.

The principal goal of an NOS is to organize a collection of distributed resources under the control of a single, integrated system. Thus, an NOS designer must, at the very beginning, determine how these resources are to be portrayed to users and to programs. A sound principle from a human engineering point of view is to provide a uniform interface to similar resources, regardless of the actual site of implementation. Uniformity may include requirements that similar resources have similar properties as well as similar patterns of access. For example, a user should be able to copy local as well as remote files within the distributed file system using a common command. Additionally, if programs can access attributes of a file, such as its date of creation, the principle of uniformity suggests that those attributes be maintained for all files supported throughout the NOS.

For a base level NOS implementation, such uniformity is not difficult to achieve, since the principal paths to all system resources accessible to users are, in fact, implemented as part of the NOS development effort. For a homogeneous meta-system NOS uniformity can often be the by product of the homogeneity. Total uniformity of the resources within a heterogeneous meta-system NOS is not achievable in general, nor is it necessarily desirable. To some extent heterogeneous systems are attractive because of the diversity of services and resources available. Uniformity of access is still, however, much needed to achieve an integrated facility. A key decision then, during the design of the system, is to determine the extent of providing or enforcing uniformity, or alternatively, determining the point at which the local host operating system conventions should become visible at the user or program interface. A designer may make separate and different decisions for the user and programming interfaces depending on the intended use of the system and the perceived sophistication of the various user populations.

A design issue which is somewhat related to uniform accessibility is the degree to which distributed aspects of the architecture should be visible to users and programs. At one extreme is complete invisibility, where a user need not be aware of network operations being performed on his behalf, nor is he able to exercise direct control over the use of distributed resources. At the other extreme is complete network visibility where the user is aware of the network and its constituent hosts, and must

directly assert control over the selection of resources to service his requests. Clearly, a system which does not provide a uniform interface cannot be totally transparent. Nor can a system that requires that explicit physical location information be provided as part of user requests be totally transparent.

It is often true that a large degree of network transparency is desirable. In many cases, a user is unconcerned with the specific combination of resources required to support his requests. This is true when resource location is not a major factor from a functional or performance viewpoint. Indeed, a goal of certain types of NOSs is to make "optimal" selections of resources. If an NOS can automatically make the selection to a user's satisfaction, then the level of detail that must be mastered to use the NOS in an effective manner is greatly reduced. However, a number of factors suggest that, at least given the current state of the art, a more visible approach is often appropriate. In currently available systems, performance frequently decreases when access to remote resources or agents is required. Furthermore, certain NOS operations may be supported only for entities which reside on the same host. When these or similar factors are present, the most appropriate system design may be to provide users with mechanisms to influence, if not specify, resource patterns. In many cases such mechanisms may be used at the user's option. A user could choose to use them or not based on his level of sophistication and his desire to obtain optimal performance. When resource selection patterns are not

specified by the user, the system would employ its own selection mechanisms, which may be adequate in many situations.

Performance considerations should also be a factor in determining the degree to which one should attempt to build a "reliable" system. Reliability measures are expensive in terms of the system resources they consume and can have an appreciable impact on system performance. A system designer must determine the amount of system resource, which might otherwise be used to support normal system operation, to be allocated to mechanisms that prevent, detect, and recover from operational malfunctions. In mission oriented systems the reliability requirement is often explicit and stated in terms quite removed from the automation of the task. With general purpose systems the requirements are frequently less rigid.

The degree to which NOS designs should take advantage of the physical distribution of the hardware resources to achieve reliable operation is unclear. For example, replication of critical system data bases can allow continued operation in the presence of localized failures, but may also introduce delays for processes using the data due to the synchronization required to correctly maintain the data. Before selecting reliability mechanisms the NOS designer needs to anticipate the critical system resources for the potential user community, and to determine the degree of reliability required by the users.

Performance in the response time sense is not the only consideration in determining the level of effort to be expended on error detection and recovery. Even when reliability mechanisms can be handled in a completely parallel fashion, thereby minimizing delay, there is still the question of overall cost. The amount of extra hardware and processing capacity that can be spent for reliability mechanisms is a fundamental system design issue.

Techniques for achieving "reliable" computing facilities are currently a research topic. When the results from this research are available, the NOS designer must decide which techniques are appropriate for various aspects of the system. For a general purpose system it may be important to maintain flexibility enabling user groups to select various parameters among the performance/cost/reliability tradeoffs in order to configure a system tailored to their needs. Whether more than one such configuration can coexist within a single NOS is an interesting question.

In addition to the general design issues discussed above, there are issues concerning the implementation of an NOS. An interesting and important one concerns the placement of control functions among the participating hosts. In developing a large computer system, a functional decomposition of the system into smaller units is almost always required. The distributed nature of an NOS adds a new dimension to the decomposition problem.

There often is a great deal of freedom in selecting the site or sites to provide a given function or maintain the data necessary to support the function. When the support for a function is centralized, the impact of the distributed environment is minimized and implementation can be simple and straightforward. There is no confusion about where to obtain the function; there need not be implementation efforts for different host types; no synchronization needs to occur; and uniformity is easy to achieve.

Two potential drawbacks to a centralized approach are: the system is vulnerable to failures of the site supporting the centralized control function; and the possibility of degraded performance exists because of the potential bottleneck represented by the single implementation and because of the necessity for interhost communication each time the function is needed. The vulnerability and bottleneck problems can be partially reduced by replicating the function at an appropriate subset of the interconnected sites. We shall call such a configuration logically centralized but physically distributed. Such a configuration may be either easy or extremely difficult to implement, depending upon the interhost synchronization needed to correctly support the function, and the number of implementations for different hosts that are required. Whenever logical centralization is used, it is done so at the expense of host autonomy with regard to the particular control function.

An alternative to logically centralized control is distributed control. For distributed control each entity needing a control function has its own implementation of the function. Typically the control module at a host cooperates with similar control modules at other hosts to accomplish the control function. The amount of interhost cooperation required depends upon the particular function and the details of the implementation, and can range from none to a large amount. Distributed control provides a means for breaking the dependence of one host on another, and additionally can result in performance improvements because of the local implementation. However, unless care is taken in employing distributed control, the costs in designing (one time cost), implementing (one time cost) and operating (continuous cost) an NOS system based on distributed control may be significantly greater than those for one based on centralized control. Thus, the system designer must decide which system functions are most appropriate for distribution and which should be centralized. Distributed control mechanisms are topics of current research. Greater understanding of them is required before they can be routinely used as the basis for an operational NOS.

In addition to NOS design characteristics which tend to support an integrated view of the distributed system components, there are operational and administrative factors which may simplify use and maintenance of the system. The feasibility and desirability of centralizing network monitoring and maintenance

functions have already been demonstrated [73]. It is important for the operational success of the NOS concept to minimize the administrative requirements for utilizing the distributed resources. A centralized administration for an NOS can potentially provide far more effective user support than can an organization which distributes administrative responsibility in a way that requires each host to individually manage its own resources. A centralized administration to support account establishment, billing, and programmer and software assistance can contribute to the overall image of an NOS as an integrated information processing utility. However, in certain environments it may not be feasible to support this type of centralized management. The current ARPANET environment seems to be an example here. In these cases there is a need for increased user sophistication to handle the administrative non-transparency of the network host configurations.

3.6 Design Issues: Resource Characteristics

In this section we focus on some of the design issues which relate to the major classes of NOS resources. For purposes of discussion, we shall divide the resources into three categories: files, processes and devices. This division derives from the conventional view of the role of a programming system as that of supporting programs executing as processes against data stored in files and accessible via devices. This simple model of programming activity appears well suited to the meta-system NOS.

Additionally, the issues regarding the functional appearance of these resources apply equally to many possible base level NOS approaches. In such systems individual host files, processes and devices, along with any necessary extensions, form the building blocks for the NOS.

3.6.1 Files as Network Resources

All but the most primitive NOS designs incorporate a unified facility for storing and retrieving data objects; that is, a networkwide distributed file system. The scheme for naming the stored objects is an important design issue for a distributed file system. A number of questions must be answered. Should users' names for objects be based on the object's physical location, or should there be a logical name space? Does a logical relationship among files also necessarily imply a physical relationship among their locations? Clearly, naming conventions for a system which makes its distribution invisible cannot include physical location. If network host specification is not part of the user supplied name, the system must locate the object. Catalogue data must be maintained to support the filename look up. The NOS models described (RSEXEC, NSW, and ELAN/PC-ELAN) take somewhat different approaches to file naming and the maintenance of catalogue data.

File data can be physically transported among the constituent NOS hosts. Heterogeneous systems which support such movement must address the problem of data translation in order to

make the data usable on different hosts. There are degrees to which the file translation problem can be addressed; the more intensive efforts attempt to support a greater degree of file interchangeability among the hosts. Related issues concern the granularity of reference (size of the accessible storage object) and the degree to which the NOS develops support for sub-file structure. The issues relating to sub-file access are beyond the scope of this report but are outlined briefly in a previous study [24]. For the file system models presented in this report, the emphasis is on the logical and physical structure that make up the distributed file system, and the mechanics of handling file access.

A network orientation also adds two new dimensions to file system development. First, some conventional notions of file sharing may need reexamination due to the physically distributed nature of the accessing agents. Second, the physical distribution can support and indeed may mandate multiple images of critical data items for reliable and expedient access. Both issues are discussed, as appropriate, within the context of the individual models.

3.6.2 Programs as Network Resources

A similar set of issues relate to the execution of programs as processes in an NOS. Important questions here include the following. How are these processes to be named? Must the user

be the source of all job execution and process creation requests, or can he have programs spawn other processes to execute other programs? What is the logical relationship among processes? How does physical location impact that relationship? Who or what (the user or some module of the NOS) selects the location for process execution and on what basis? Is process location fixed after the process is created, or may processes migrate? What is the control structure between distributed processes, and how is it supported? If the NOS does not provide application program support for dynamic process creation and interprocess communication, are there mechanisms outside of the NOS for easily accomplishing these functions? What, if any, are the relationships between an NOS process and processes within the constituent host operating systems?

Some NOS systems may support the concept of network oriented processes only to the extent that it can be used within the implementation of the NOS itself. Others may be more ambitious and attempt to support a network process structure at the application level. Facilities for supporting network oriented processes are now only in the experimental stages. A key problem is defining a job structure which has widespread utility and which can be efficiently implemented. Some of the models to be discussed address this problem to varying degrees.

3.6.3 Devices as Network Resources

Special hardware devices usually are an important part of a large scale computer utility. The same can be expected to be true for an NOS which additionally introduces the possibility of sharing expensive or underutilized hardware devices among physically distributed users. Indeed, convenient access to such devices can be a major motivation for building an NOS.

Some devices, such as line printers, may have a number of instances within the facility. However, because of the importance to the user of their physical proximity, not all of these devices may be equally usable. Therefore, a user needs the ability to force the use of a particular convenient device. The system can often arrive at an appropriate selection based on information it maintains about the user and the nature of the device. However, to achieve a greater degree of flexibility, system support for user controllable factors which influence the resource selection decision are often appropriate, provided they are compatible with the desired degree of network transparency for the overall NOS.

Except for some ad hoc attempts to support specific devices within specific system contexts, the issue of uniform handling for remote devices has received little attention by NOS researchers. A major problem is providing uniform access paths to the various device types and instances. Should device access be through "virtual" devices for which the NOS assumes

responsibility for mapping between virtual and physical device characteristics? Or alternatively, should access to a remote device be supported by a direct communication path from the accessing process to the actual software "driver" for the device on the host that supports it? Other approaches are also possible. Questions relating to devices have only been superficially addressed in the context of existing NOS models, and are difficult to deal with in the abstract.

3.6.4 Network Access Control and Resource Allocation

The issues surrounding NOS resources, as discussed above, cannot be easily separated from the related issues of access control and resource management. Below we list some key questions in these areas which should be asked when classifying an NOS design.

Regarding the approach to access control: Is there a single, unified mechanism, or are there multiple mechanisms based on those already present in the individual host operating systems? Is there systemwide uniformity to the types of resources to which access is controlled and to the modes of access permitted? An important part of any access control system is authentication. For NOS access control, what entities are authenticated, and who performs the authentication? How often must the authentication take place (e.g., once per session, for each request), and how are previously authenticated entities identified.

Regarding resource management, an important question is who makes the resource selection: the user, the system, or a combination of the two? Different approaches to the problem of handling resource management within an NOS can be further characterized by the information that contributes to the selection process. Is it a dynamic decision based, perhaps, on current network conditions, or is it a static one based, perhaps, on information in a user profile? Another issue of interest is the level of NOS resource management. At what point does the NOS relinquish control to the local host mechanisms for resource management?

3.7 Summary

The issues raised in this section have tried to set a framework for assessing various characteristics of the NOS models described in Section 4. Of course, not all of the systems to be described address all of the issues. These issues will also provide a framework within which the NOS system models can be compared and contrasted in terms of the functionality they support and the techniques used to implement them.

4. NOS Models - Functional Description

This section describes the NOS models investigated as part of this study. The models vary in sophistication. We believe each is well suited for some situation or set of constraints. Some of the system models described were developed as part of this study. For others, the systems were developed elsewhere, but their NOS models were developed as part of the study. All but the first system described represent attempts at "total" NOS systems in the sense that they include both command language interpreters and program execution environments.

The NOS models are presented in order of increasing sophistication. The models to be described are:

- Automated TELNET and FTP (ATF).

This model was developed as part of the study. As the name suggests, ATF is a system that automates the procedures required to gain terminal access to remote systems and to move files between systems.

- Resource Sharing Executive (RSEEXEC).

RSEEXEC is an existing NOS system developed by BBN. It is primarily a homogeneous system for the PDP-10 hosts on the ARPANET which run under TENEX or TOPS-20. RSEEXEC supports a distributed file system at the command language level and at the program execution level.

- National Software Works (NSW).

The NSW is an existing system being developed by a team of contractors which includes BBN. NSW is an operating system for a heterogeneous collection of hosts. It implements a distributed file system, and provides a sophisticated command language interpreter and a network program execution environment.

- ELAN.

ELAN is an NOS model developed as part of this study. It is designed for heterogeneous collections of hosts. The ELAN design is based in part upon experiences with the RSEXEC and NSW systems, and it addresses some of the deficiencies of those systems.

- PC-ELAN.

PC-ELAN was developed as part of this study. It extends the ELAN NOS design to support personal computers. It addresses the problem of integrating personal computers into an NOS.

Before describing these models in detail, we state below some assumptions we have made regarding the environment in which these systems would operate. These assumptions serve to constrain the type of NOS system under consideration.

- Network.

We assume that a wide variety of host types are connected by a general purpose computer network. For our purpose the underlying communication discipline used, e.g., packet switching or circuit switching, is not important. When it is necessary to be more specific about network characteristics, we shall use the ARPANET as our model.

- Hosts.

A heterogeneous collection of hosts is assumed. For some host types there may be more than a single instance. There is a large investment in existing host hardware. It is assumed that the hardware can not be extensively modified (enhanced or replaced) to support an NOS. The existing host hardware must be part of the NOS.

- Host Software.

There is a large investment in host software. This investment is both at the application level and the system level. Little modification to host operating system software is permitted. Any modification that occurs must be as additions to, rather than changes or removal of, functionality. There are two reasons for this: because of the large number of host types and the effort required to develop a host operating system, it is infeasible to develop

new operating systems for all the hosts in order to support NOS activity; it is important that application software continue to be able to execute - any changes to the abstract machine seen by application software must be upward compatible.

- Users and Applications.

An NOS must support general purpose computing and a wide range of applications. It must support turn key users and application programmers, as well as system programmers for the constituent host operating systems. It is desirable for applications to make use of the distributed resources. In addition to allowing a user to run programs on different hosts, the NOS should support scenarios for which the input to one program is the output from another. Thus, the NOS must support the movement of files from host to host, either at the explicit request of a user or his program, or implicitly as the result of a data reference operation by the user or his program. In addition, some sophisticated applications may be multi-computer themselves. Therefore, the NOS should support the implementation and operation of distributed user processes.

- Host-to-Host Communication.

Interhost communication is significantly more costly than intrahost communication in terms of delay and the computational resources required to support it. Consequently, interhost communication is likely to be a performance bottleneck for an NOS. Users are likely to see significantly different responsiveness characteristics for seemingly similar activities when some involve extensive host-to-host communication and others do not.

4.1 A System for Automated TELNET and FTP

4.1.1 ATF System Objectives

This section describes a relatively modest system, but one which we believe can be used quite effectively by sophisticated users. The design of this system had three principal constraints:

- Its implementation must require only a modest effort.
- It is not permissible to modify the resource bearing hosts in any way.
- The hosts must maintain a high degree of autonomy.

It is assumed that the constituent resource bearing hosts implement the TELNET [1] and FTP [3] protocols. That is, TELNET and FTP server processes reside on all of the hosts. This makes it possible to remotely login to each host, and to move files to and from each host.

The system to be described is a system for automated TELNET and FTP (ATF). It performs many of the routine actions users must normally perform to initiate processes on remote hosts and to move files from one host to another. By automating these actions an ATF system eliminates much of the tedium and many of the chances for error that normally occur in accessing remote network resources.

We believe that ATF systems are best suited for relatively sophisticated users. For such users, we believe that an ATF

system, while it does not substantially change the nature or power of the services available on the resource bearing hosts, does make them significantly easier to access and to use.

4.1.2 ATF System Features

In its simplest realization an ATF system would provide two functions: a job initiating function which would enable a user to start a task running on a specified host; and a file movement function which would enable a user to move a file from a specified file storage area on one host to another specified storage area on another host. To be more specific, we shall call these two functions START-JOB and MOVE-FILE.

Both functions would be automated in the sense that the user would need to specify for START-JOB only the host for the desired job, and for MOVE-FILE only the names of the source and destination files. The ATF system would perform all the steps necessary to accomplish the requested actions including: connecting to and communicating with TELNET or FTP server processes on the remote hosts, supplying the necessary access control information, initiating file transfers, and so forth.

To support these actions the ATF system would maintain information about each user, such as a login name, password and account necessary to access each host he commonly uses. This and perhaps other information about the user, such as the characteristics of the terminal he commonly uses or the name of a

program to be run at a given host each time a job is initiated there, would be maintained by the ATF system in a user profile.

The information in a user's profile must be supplied by the user. However, after he has provided it, he need not re-supply it each time ATF needs it. To acquire this information, an ATF system could either provide a profile editing function, which could be used to enter information into a user's profile; or it could interrogate the user if the information required to satisfy a START-JOB or MOVE-FILE command was not already in the profile; or it could do both. To be specific, we shall assume a profile editing function which the user can invoke via a command called EDIT-PROFILE.

During the course of a session with an ATF system a user would typically initiate and terminate several remote jobs, and might wish to have several jobs active simultaneously. To allow this, ATF must be capable of simultaneously supporting multiple TELNET connections. In addition, to permit the user to distinguish among these jobs, ATF associates a name supplied by the user with each, and provides means for the user to switch his attention among them. To be specific, we shall assume an ATF command called USE which can be invoked when a user wants to direct his attention to a specified job, and a special control character (CONTROL-Z) which can be used to remove his attention from the current job and redirect it to the ATF command interpreter.

To explain the operation of an ATF system in more detail we shall describe the actions it would take to satisfy a typical sequence of user commands. To do so, we shall use a hypothetical ATF command language.

The commands for this language are as follows. Commands are entered by typing the command name, followed by command parameters (if any), and terminated by a <carriage-return>. Optional parameters may be specified by "subcommands"; to enter subcommand mode the command line is terminated by the sequence <comma><carriage-return>, rather than by a single <carriage-return>. The character "-" is used as a prompt character by the ATF command interpreter to signal that it is ready to accept a user command, and the two character sequence "--" is the subcommand prompt which indicates that the ATF command interpreter is ready to accept a subcommand. A "real" ATF command language would support additional features such as line editing, command recognition and completion, and on-line help. Since such features are unimportant to discuss the ATF concept, we leave them unspecified in our hypothetical command language. However, we would expect an implementation of an ATF system to support such functions. In the examples that follow, user input will be underlined to distinguish it from system response. In addition, certain "noise words" which would not necessarily be part of the user-system dialogue are included to assist the reader in understanding the interactions; these noise words are parenthesized.

The sequence of commands to be described are ones a hypothetical programmer might use as part of a debugging session. We assume that he needs to modify a COBOL source program called TEST stored at host A, and then to compile the program using an optimizing COBOL compiler that runs on host B, and finally to load the resulting object module with other object modules created by other programmers that are stored on host C. We shall assume that his user profile already includes information for hosts A and C, but that it does not yet include the necessary information for host B. The programmer's name is Smith.

After invoking the ATF system, the user could initiate his editing activity by the command:

-START-JOB (named) J1 (at host) A
Job J1 successfully started at host A

To start job J1 ATF would establish a connection with the TELNET server process at host A. It would read Smith's profile to obtain his login name, password and account for host A, and then it would send them as part of the login command appropriate for host A.

At this point the user is free to issue other ATF commands. To do his editing he would type:

```
-USE (job) J1  
@RUN EDITOR  
Input File: TEST.COBO  
.  
  (user edits file)  
.  
Output File: TEST.COBO  
@  
<CONTROL-Z>
```

The USE command instructs ATF to direct subsequent user terminal input to job J1 at host A. The lines that follow it are the user's interactions with the text editor at host A. These interactions follow the conventions of host A and the particular text editor being used. The character "@" is host A's prompt character. The <CONTROL-Z> redirects the user's input from job J1 to the ATF command interpreter.

Before Smith can compile his program at host B he must augment his profile. The following interactions modify the profile:

```
-EDIT-PROFILE  
*ADD-ENTRY  
Host: B  
Login Name: RSmith  
Account: 127  
Password: WXYZ  
RSmith at host B, account 127 added to your profile.  
*PRINT-PROFILE  
Smith at host A, account 10530  
RSmith at host B, account 127  
RTS at host C, account 11320  
PROJECT-1 at host C, account 220100  
*QUIT  
-
```

Here "*" is the prompt character used by the profile editor. The ADD-ENTRY function is used to supply the information necessary to enable ATF to initiate tasks for him at host B. Smith also prints his profile. As a result of the addition made, the profile contains four entries, two of which are for host C.

To compile the program the user would issue the following sequence of commands:

```
-MOVE-FILE (file) TEST.COBOL (from) A (to) B  
TEST.COBOL moved to host B  
-START-JOB (named) J2 (at host) B ,  
-RUN (program) COBOL  
Job J2, Program COBOL successfully started at host B  
-USE (job) J2  
Input File: TEST.COBOL  
Output File: TEST.OBJECT  
Compilation completed  
No Errors Detected  
:  
<CONTROL-Z>
```

To satisfy the MOVE-FILE command ATF must establish communication with FTP server processes at hosts A and B. In addition, ATF must determine the "complete" names for both source and destination files. ATF can notice that the user profile contains only a single login name for hosts A and B, from which it can assume that the source file resides in the Smith directory at host A and that the intended destination for the file is the RSmith directory at host B, and furthermore that the use wishes the file copy at host B to retain the name TEST.COBOL. If the user wanted to move a file from a different source directory, or to a different destination directory, or to have a different name for the destination file, he could use subcommands to so instruct ATF.

A subcommand is supplied as part of the START-JOB to instruct ATF to start the COBOL compiler at host B. After ATF creates the J2 job for the user it sends the appropriate host B command to run the COBOL compiler. The character ":" is host B's prompt character. After the compilation is completed the user returns to the ATF command interpreter. If the user had had other tasks to do during the compilation at host B, he could have redirected his attention back to ATF and from there to the other tasks.

Finally, to move the compiled program to host C, the user might use the command:

```
-MOVE-FILE (file) TEST.OBJECT (from) B (to) C ,  
--DESTINATION-FILE-DIRECTORY PROJECT-1  
--DESTINATION-FILE-NAME LOADMODULES>TEST.RELOCATABLE  
--DELETE-SOURCE-FILE
```

Since the user profile contains two login names for host C, the user instructs ATF that the file should be moved into the directory associated with PROJECT-1. A sophisticated ATF system might allow a user to distinguish between "primary" and "secondary" file directories for various hosts in his profile and would select the primary directory as a default in file movement operations. In addition, it might enable the user to specify for the duration of a session, or until re-specified, a default directory for file movements involving a particular host. The second and third subcommands instruct ATF that the destination file is to be named LOADMODULES>TEST.RELOCATABLE rather than TEST.OBJECT, and that after it has been moved, the source file

should be deleted. Notice that the file name syntax for hosts B and C differ. The user must deal with these differences himself. As with the previous MOVE-FILE command, ATF supplies FTP protocol commands to FTP server processes at the appropriate hosts to accomplish the file movement and subsequent deletion.

The above discussion has served to illustrate the features a rudimentary ATF system might provide and the manner in which they could be used. To summarize, the basic characteristics of an ATF system are:

- User specific information needed to automate TELNET and FTP, such as login names and passwords, is acquired by ATF and maintained in a user profile.
- The user manually initiates all job start up (TELNET) and file movement (FTP) operations specifying the appropriate host and file names.
- ATF derives the name and password parameters appropriate for the specified hosts from the user profile, and for FTP operations, possibly the complete names of the files.
- ATF performs all the network access, login, process initiation, and FTP operations required to accomplish job start up and file movement.

There are a number of ways a rudimentary ATF system might be made more sophisticated to provide a higher level of service to its users. These include:

- Implementation of additional file operations.

The ATF system described above supports only file movement. Users typically require other file maintenance operations, such as the ability to rename, copy, append and delete

files. For the ATF system described, to delete a file at a remote site the user would have to start a job at the site and then deal directly with the command language interpreter at that site to delete the file. For example:

```
-START-JOB (named) A (at host) B  
  Job A started at host B  
-USE (job) A  
  
@DELETE C.D  
@LOGOUT  
  
<CONTROL-Z>  
  
-TERMINATE (job) A
```

It would not be very difficult to extend our ATF system to support routine file maintenance operations. For example ATF could support a DELETE operation:

```
-DELETE (file) C.D (at host) B  
  File C.D at Host B deleted
```

As in the case of file movement, ATF would interact with the FTP server process at host B to accomplish the deletion, obtaining the necessary access control data for host B from the user's profile. ATF operations for rename, copy, append and so forth could be defined and supported in a similar fashion.

- Background ATF actions.

For the system described above, all ATF operations occur in the "foreground" in the sense that every operation occurs while the user waits. Thus, the user is prevented from

AD-A056 078

BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS
NETWORK OPERATING SYSTEMS.(U)

F/G 9/2

UNCLASSIFIED

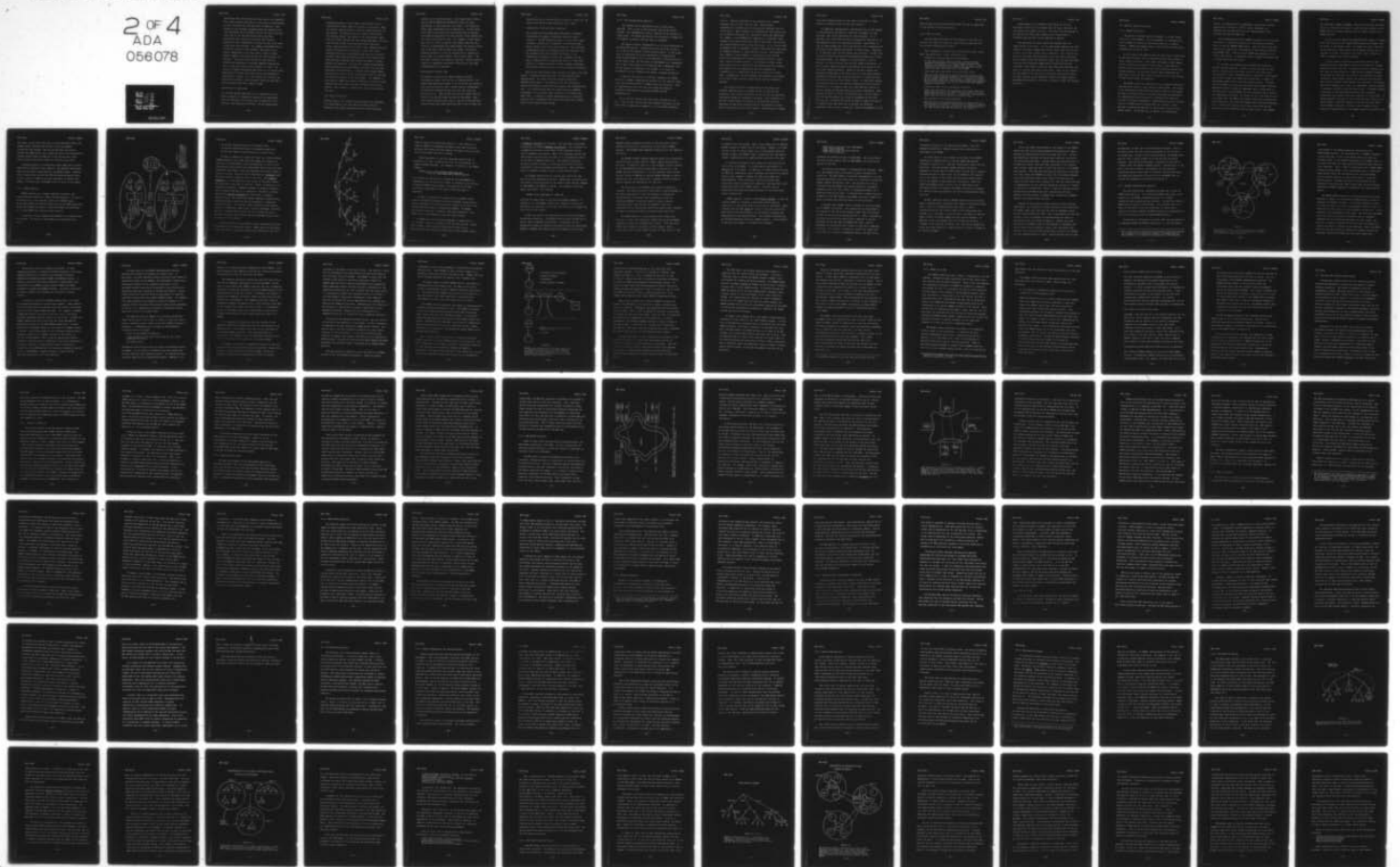
MAY 78 R H THOMAS, R E SCHANTZ, H C FORSDICK

F30602-76-C-0425

RADC-TR-78-117

NL

2 OF 4
ADA
056078



performing other operations until the current one completes. This could be quite annoying to a user when it takes minutes or tens of minutes for the operation to complete. Such delays are not uncommon, depending upon the operation and load conditions for the network and for the hosts involved. Since none of the ATF operations require any user interaction after they are initiated, they could be performed in the "background", freeing the user to do other things while they proceed. For example, when MOVE-FILE is invoked to initiate a file movement, ATF could return control immediately to the user and be ready to accept another command, informing him that the file transfer was underway. ATF could either inform the user when the transfer was completed, or provide commands enabling the user to query the status of the transfer, or both. The ability to support background operations is, of course, most beneficial for longer operations such as the movement of large files. However, it is also useful for job start up operations and shorter file transfers which can take up to several minutes when host loads are high.

- Persistence in operations.

For the ATF systems described so far, should one of the hosts necessary to complete an operation be inaccessible when the operation is initiated (due to network or host failure), the operation would fail. An ATF system which

showed persistence in such cases could provide users a significantly higher level of service in such cases. There are several ways persistence could be incorporated into an ATF system. The system could, at a user's instruction, continuously monitor the status of an inaccessible remote host and notify him when the host became available. At that time the user could re-initiate the failed operation. Another approach would be for ATF to take responsibility for completing operations initiated by a user even if the necessary hosts were inaccessible when they are initiated. As above, ATF would persistently monitor the status of the required hosts, and when they became available, it would perform the operation. An even more sophisticated system would continue in its persistence even after the user had terminated his ATF session. A system that provides highly automated persistence should also provide means for the user to control the extent of the persistence. For example, in some cases it might make sense to continue to work on a file movement operation after the user terminates his on-line session, and in others, it might not; only the user knows for sure.

- File syntax conversion.

Although many of the routine file operations are automated to a large degree, the user must still deal with the different file naming conventions of the constituent host

systems on an individual basis. Even among similar hosts, such as PDP-10 TENEX and DEC TOPS-20, there are minor differences in the syntax for file names. For dissimilar hosts these differences can be substantial. An ATF system could help a user deal with these differences by mapping file names he specifies into the syntactic forms appropriate for the various hosts he uses. For example, an ATF system that runs on a host of type A could embody some heuristic rules for transforming file names between the syntax used by host A and those used by other host types. This would simplify things for a user by allowing him to specify all files names in a single syntax, that of host type A. The ATF would transform the names as required. Further study is required to determine the extent to which such file name transformations are feasible.

- Simultaneous multiple jobs.

The degree to which an ATF system supports multiple simultaneous remote jobs can vary in sophistication. The system described above allows a user to switch his attention back and forth between jobs. Output from an "unattended" job should be buffered by ATF until the user turns his attention to it. When he does so, the user should have the option of having the output printed or discarded. Other user features would allow a user to ask to be notified when output arrives from some particular job or set of jobs, and

would allow him to instruct ATF to record in a data file the output received from some particular job or jobs.

- Integration of ATF and local resources.

ATF systems have been described principally as command interpreters to support convenient access to remote resources. An ATF system that runs on a host that itself includes a collection of resources, could also provide direct access to the local host resources. This could be accomplished either by integrating the ATF functions into the standard local host command interpreter, or by integrating the important local host commands into ATF. Such a system would be tremendously useful to local host users for it would give them a single command interpreter that provides uniform access to local and remote resources.

Each of the items listed above increases the power of an ATF system. A system incorporating all of them would be quite sophisticated. Of course each item requires additional implementation effort. The designer of an ATF system must balance the benefit of each such item against the implementation cost. It is worth noting that the items above are somewhat independent of one another, and a designer could choose from among them. In addition, phased implementations are possible, starting with a rudimentary ATF system and advancing in steps toward a more sophisticated system.

4.1.3 ATF Implementation Approach

ATF systems can be implemented using a single agent approach. This is one of the principal attractions of ATF systems - the implementation effort required can be limited to a single host. (As stated previously, the existence of TELNET and FTP modules on resource bearing hosts is assumed.)

ATF agents could be implemented to run on hosts dedicated to providing ATF service. Alternatively they would be written to run on one of the network resource bearing hosts. In the former case, an ATF system could be considered as a sophisticated network access machine of the sort described in Sections 2 and 3. In the latter case, the ATF could be implemented as a "subsystem" that a user could invoke in the same way other subsystems, such as text editors or compilers, are invoked, or it could be integrated into the standard host command language interpreter.

To provide its services an ATF must implement the TELNET protocol to support communication between its user and his remote jobs, and the FTP protocol to support user file activity. From an implementation point of view, two things distinguish a rudimentary ATF system from standard TELNET and FTP implementations.

First, the ATF creates jobs and starts programs for the user. Thus it must interact with the command interpreter of the remote host, transmitting commands to it and reading responses

from it. Specific knowledge of the different host command languages must be built into the ATF. The principal implementation problem here is interpreting responses from the remote hosts. These responses are intended for human users, and are frequently difficult for a program to parse and understand. For example, on the ARPANET the response to a successful login command varies greatly from host type to host type, and can even vary among different hosts of the same type. It is a non-trivial programming task to implement an ATF to understand the wide range of command responses it might possibly receive. The situation with respect to file transfer is much easier since the FTP protocol was designed as a process-to-process protocol, rather than a user-to-process protocol. The concept was that a user would interact with an FTP "user" process which would transform his requests into the appropriate protocol commands for transmission to a cooperating FTP server process on a remote host. Consequently, the FTP commands and responses are standard for all host types, and are easily parsed and interpreted by programs.

The second thing that distinguishes an ATF system from standard implementations of TELNET and FTP is that the ATF maintains a user profile from which it derives the user specific information needed to accomplish its tasks. The information in the user profile makes it possible to provide ATF services. Typically, an ATF would maintain profile information in a data base or file stored on its local system. There might be a single

file which holds profiles for all users of the ATF, or there could be a separate file for each user profile.

An important implementation consideration for an ATF system is the protection provided for sensitive information in user profiles, such as the passwords for remote systems. It is generally not a good idea to store the passwords required for an ATF system on-line in an unprotected fashion. The problem is that an unprincipled user could read another user's passwords and use them to gain unauthorized access to remote hosts. Therefore it is important that the ATF protect user profiles. An ATF could do this by using the protection mechanisms provided by its local operating system. For example, it could maintain a user's profile in a file to which only the user himself had access. If this is done, sensitive profile information is as secure as the local host operating system. Should an intruder subvert the protection mechanisms of the local host, he could read sensitive information from ATF user profiles and use it to access private information stored on other network hosts. An additional measure of protection could be achieved by encrypting sensitive profile information. For example, each user might specify a key to be used to encrypt and decrypt sensitive profile information. This information would be stored in encrypted form. The key itself would be stored nowhere, but would be remembered by the user. When the user invoked ATF he would be asked to supply his key as a password which ATF would use to decrypt the sensitive information into a form it could use. Recent developments in the

area of public key encryption mechanisms [54] may be applicable to the problem of profile security.

4.1.4 ATF as an NOS

The principal attraction of ATF systems is that they can provide very effective means for using network resources with only minimal implementation effort.

Their fundamental limitations all derive from their single agent implementations. These limitations are:

- Absence of a program execution environment.

An ATF system cannot easily provide a network program execution environment for remote programs because NOS software is required in the remote hosts to do so. Such software would violate the single agent approach and the constraint that host systems not be changed.

- Users must deal with hosts on an individual basis.

The user must establish an account with each network host and be issued a name and password valid for the host before he can access resources on the host. In addition, he must deal with the different resource naming conventions employed by the various hosts, such as the file naming and command language conventions.

- Users must make resource management decisions.

Users must make their own decisions in the sense that they select where files are to reside and when files are to be moved from one host to another. In addition, they determine which hosts are to be used to run various jobs.

- Data access is a two step operation.

Because there is no program execution environment and all data movement is explicit, data access, in general, requires two steps: the data must be moved to the accessing host by the user before a process on the host can access it.

Enhancements to a rudimentary ATF system of the sort described in Section 4.1.2 address some of these limitation and can reduce their impact on users. The other four NOS systems to be described in this report address these limitations in different ways and with various degrees of success.

The importance of these limitations depends upon the capabilities of the users. For an experienced computer user they might be relatively unimportant and the direct user control may, in fact, be an asset rather than a liability. For a novice turn key user they might prevent effective use of network resources. With a sophisticated ATF system an expert user can make extremely efficient use of network communication facilities and host resources. The reason is that the user knows precisely when and where to initiate jobs and to move files. Thus, he can instruct the ATF system to do exactly the operations required to handle his task, whereas a more "powerful" system that makes resource management decisions for its users is likely to make less optimal decisions.

4.2 Resource Sharing Executive

4.2.1 RSEEXEC Objectives

The Resource Sharing Executive (RSEEXEC) is an NOS system that runs on the ARPANET. To our knowledge, it represents the first attempt to develop an operating system for a computer network. RSEEXEC development began in 1972 and was substantially completed by the end of 1974.

At the time it was apparent that for many users an NOS would be the most effective, and for some perhaps the only, means of using network resources. Large hosts, by making a small amount of their resources available to small hosts, could help the smaller hosts provide services that would otherwise exceed their limited capacity. By sharing resources among themselves the large hosts could provide a level of service better than any one of them could provide individually.

The RSEEXEC system was conceived of as a research vehicle to experimentally explore a wide variety of NOS issues. The issues to be investigated ranged from the types of features that would be useful to network users, to system structures and mechanisms for implementing these features, to strategies for ensuring reliable, fail-soft performance. Experimentation with an initial version of RSEEXEC was encouraging and, as a result, it was decided to develop and maintain an operational version of the RSEEXEC system. The RSEEXEC was, by design, an evolutionary

system. We planned first to implement a system with limited capabilities, and then to let it evolve, expanding its capabilities, as experience with and understanding of the problems involved was acquired.

Early experimentation with RSEXEC proceeded in two principal directions: coupling the operation of large general purpose hosts, primarily ARPANET TENEX hosts; and, utilizing large host resources to support the requirements of small hosts, primarily the ARPANET TIPS. From the outset RSEXEC was designed to be a "total" NOS system. It implements both a command interpreter and a program execution environment.

At the time, the TENEX abstract machine was becoming a popular network resource. Further, we observed that for many users, in particular those whose access to the network was through TIPS or other non-TENEX hosts, it shouldn't really matter which host provided the TENEX abstract machine. A number of advantages would result from such resource sharing. The user would see TENEX as a much more accessible and reliable resource. Because he would no longer be dependent upon a single host he would be able to access TENEX resources even when one or more of the TENEX hosts were down. Of course for him to be able to do so in a useful way, the TENEX file system would have to span across host boundaries. The individual TENEX hosts would see advantages also. Due to local storage limitations, some sites did not provide all of the TENEX services to their users. For example,

one site didn't support FORTRAN. Because the services available would, in effect be the "union" of the services available on all TENEX hosts, such hosts would be able to support access to all TENEX services.

During the early design and implementation stages, it became clear that certain of the capabilities planned for RSEXEC would be useful to all network users. An example of such a capability is the ability to inquire where in the network another user is and then to "link" a terminal to the other user's terminal in order to engage in an on-line dialogue.

A large class of users with a particular need for such capabilities were those whose access to the network is through mini-hosts such as the TIP. A frequent source of complaints by TIP users was the absence of a sophisticated command language interpreter for TIPs and, as a result, the inability to obtain information about network status, the status of various hosts, and so forth, without first logging into some host. Furthermore, even after logging into a host, the information readily available is generally limited to that particular host. A command language interpreter of the type desired required more resources (especially core memory) than were available in a TIP alone. It became apparent that with a little help from one or more of the larger hosts it would be feasible to provide TIP users with a good command interpreter. Further, since a subset of the features already planned for the RSEXEC matched the needs of the

TIP users, it was clear that with a little additional effort the RSEXEC system could provide TIP users with the command interpreter they needed. The service TIP users can obtain through the RSEXEC which uses a small portion of the resources of several network hosts is superior to the service they could obtain either from the TIP itself or from any single host.

Although primarily a homogeneous system comprised of similar TENEX hosts, dissimilar hosts (Multics, IBM 360/TSO, PDP-10/ITS) have been partially integrated into the RSEXEC system. Recently hosts running under the new DEC TOPS-20 operating system, which is very similar to TENEX, have been completely integrated into RSEXEC. Currently over 40 ARPANET hosts are part of the system.

4.2.2 RSEXEC Features

RSEXEC executes as a command language interpreter and program execution monitor on TENEX and TOPS-20 hosts. It acts to provide access to the combined resources of ARPANET hosts (principally TENEX and TOPS-20 hosts) much as the TENEX operating system and command interpreter (called the EXEC) provide access to a single TENEX host resources (See Figure 6).

An important feature supported by RSEXEC is its network wide file system. The design of the RSEXEC file system had three goals:

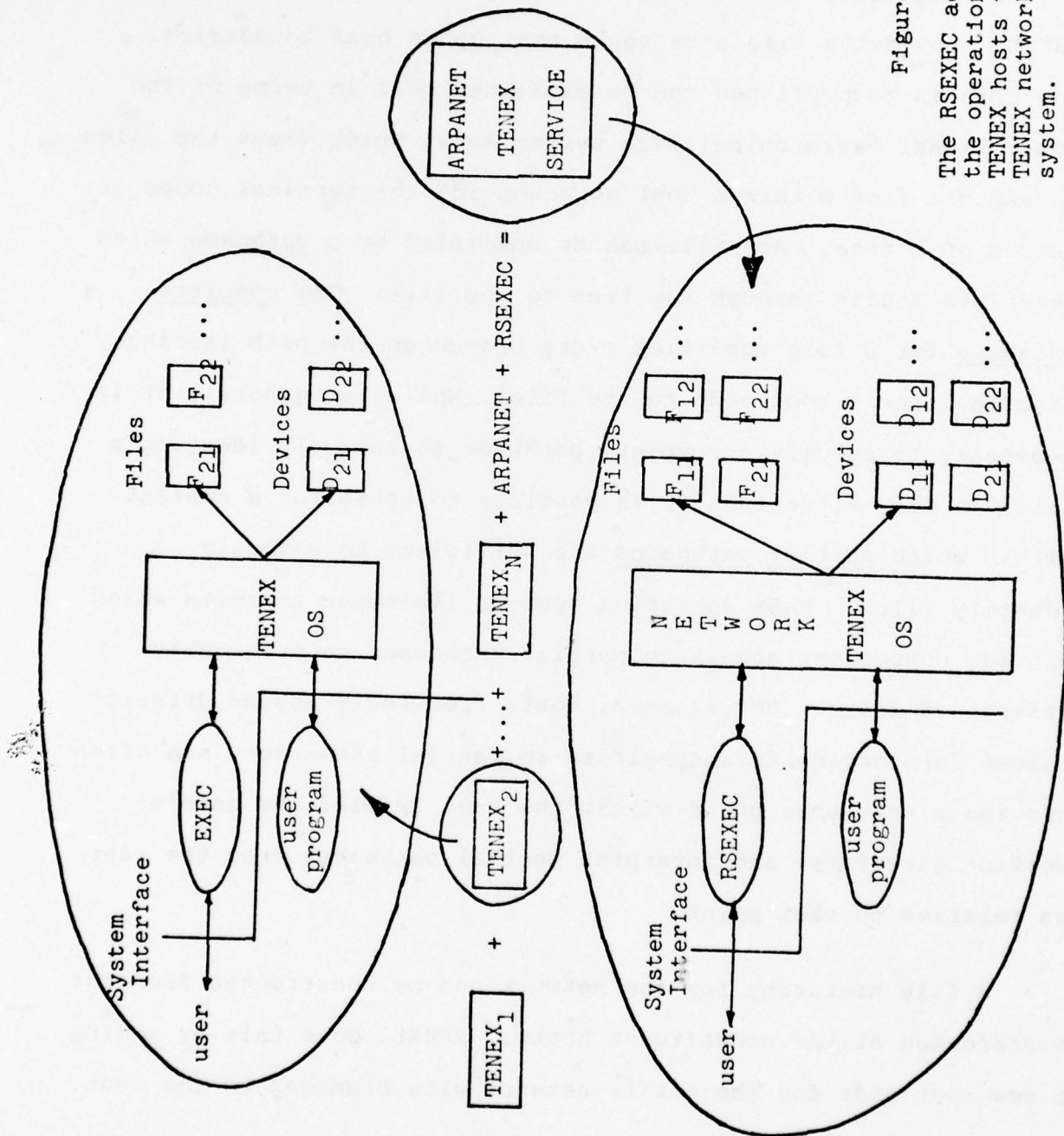


Figure 6.

The RSEXEC acts to couple the operation of ARPANET TENEX hosts to provide a TENEX network operating system.

- To provide uniform access to all network files.
- To provide convenient access to frequently used files.
- To provide highly reliable access to critical files.

To make it possible to access all files in a uniform manner, RSEXEC provides a file name space that spans host boundaries. How this is accomplished can be explained best in terms of the conventional hierarchical file system model which views the files accessible from within a host as occupying the terminal nodes or leaves of a tree. Any file can be specified by a pathname which describes a path through the tree to the file. The complete pathname for a file specifies every branch on the path leading from the tree's root node to the file. While, in general, it is necessary to specify a complete pathname to uniquely identify a file, in many situations it is possible to establish a context within which partial pathnames are sufficient to uniquely identify files. Most operating systems implement contexts which provide convenient access by partial pathnames to frequently referenced files. For example, hosts frequently assume default values for components unspecified in partial pathnames, and often provide a reference point within the tree (called the user's working directory) and interpret partial pathnames from the user as relative to that point.

A file hierarchy for the network can be constructed from the hierarchies of the constituent hosts. RSEXEC does this by adding a new root node for the entire network with branches to the root

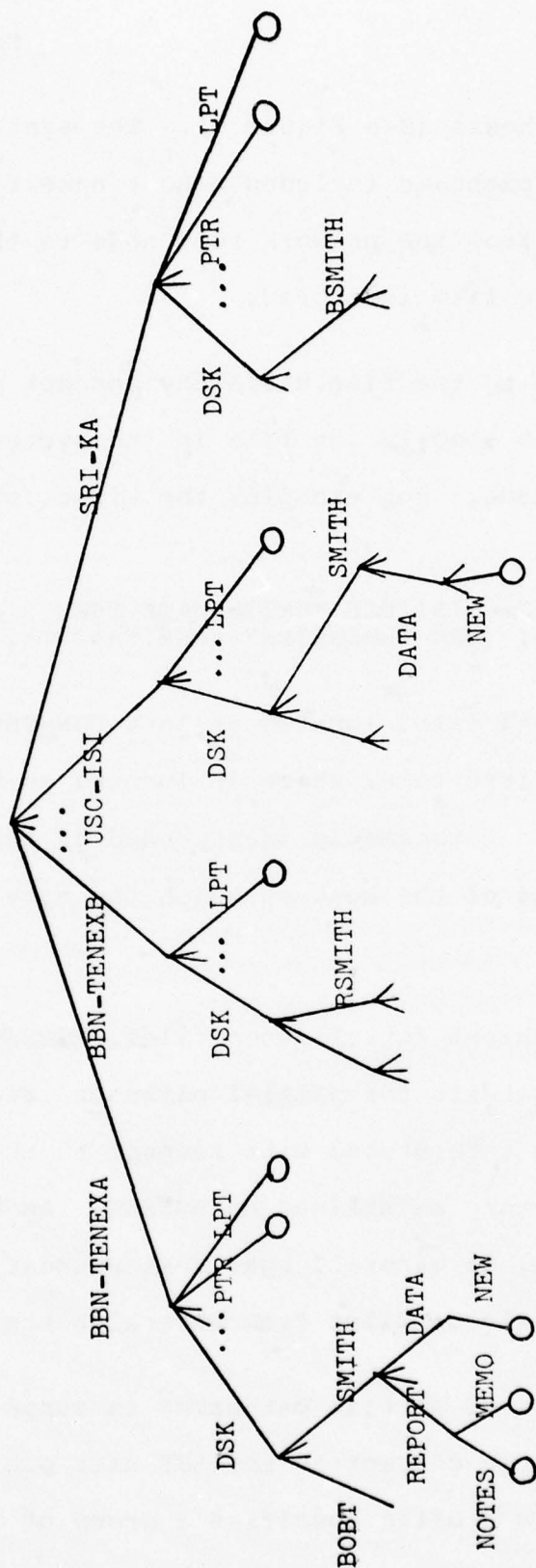


Figure 7.
RSEXEC file system hierarchy.

nodes of each of the hosts (See Figure 7). The syntax for a complete RSEXEC file pathname includes a host name field that specifies the branch from the network root node to the root node for the host where the file is stored.

These extensions to the file hierarchy concept make it possible for a user to specify any file in the system by its complete RSEXEC pathname. For example, the effect of the RSEXEC command:

```
-APPEND (file) [USC-ISI]DSK:<SMITH>DATA.NEW  
      (to file) [BBN-TENEXB]DSK:<RSMITH>DATA.OLD
```

is to modify the second file, located at host BBN-TENEXB, by appending to it the first file, which is located at host USC-ISI. A complete RSEXEC file pathname is interpreted in the same uniform way regardless of the host at which the name is generated.

To make it convenient to reference files, RSEXEC allows users to establish contexts for partial pathname interpretation. Partial pathnames are interpreted with respect to the user's own private working directory maintained by RSEXEC. An RSEXEC working directory may, in general, span host boundaries in the sense that it may catalogue files from several hosts.

In RSEXEC the use of partial pathnames is supported by a user profile, similar in concept to the ATF user profile. Among other things, a user's profile specifies a group of file directories on the constituent hosts which taken together define

a composite directory for the user. The individual constituent directories are called component directories. The "contents" of a composite directory is the union of the "contents" of the user's component directories. When a file pathname without site and directory qualification is used, it is interpreted relative to the user's composite directory. The composite directory therefore serves to define a reference point within the file hierarchy for partial pathname interpretation. That reference point is somewhat unusual in that it spans several hosts.

An example should serve to clarify the role of the user profile and the composite directory. Assume that the profile for user Smith contains directories for SMITH at host USC-ISI, RSMITH at BBN-TENEXB, and BSMITH at SRI-KA. His composite directory spans three hosts. The command:

```
-APPEND (file) DATA.NEW (to) DATA.OLD
```

achieves the same effect as the previous APPEND command. To respond to it, the RSEXEC consults the composite directory to determine the locations of the files, and then acts to append the first file to the second.

A user can control the interpretation of partial pathnames by editing his profile. The RSEXEC profile editor allows him to add or remove component directories. In addition, he can dynamically change his effective working directory by instructing RSEXEC to RELEASE and ACQUIRE specified component directories.

RELEASE removes component directories from the user's active composite directory for the duration of the current RSEXEC session; and ACQUIRE adds component directories to the composite directory.

The RSEXEC command language supports normal file maintenance operations such as copy, delete, rename, append and directory printing. In addition, there is a file movement function similar to the ATF MOVE-FILE function described in Section 4.1. The RSEXEC version of MOVE-FILE is a little different from the ATF version in that a complete or partial RSEXEC pathname is used to specify the file to be moved, and a component directory name is used to specify the destination for the file.

The third goal of the RSEXEC file system was partially realized. The objective was to allow users to take advantage of the distributed nature of the file system to increase the "accessibility" of certain files considered critical by instructing RSEXEC to maintain images of them at several different hosts. The idea was for RSEXEC to take full responsibility for maintaining these multi-image files, enabling the user to manipulate them as ordinary single copy files.

The RSEXEC supports a semi-automatic multi-image file facility. A user can create multi-image files by using RSEXEC commands to create the necessary file copies. From that point RSEXEC will attempt to maintain the file images. When a multi-image file is read, RSEXEC selects one copy which it uses

to complete the read access. When a multi-image file is updated, RSEXEC attempts to update all of the copies. However, if a host storing a copy of a multi-image file is inaccessible when the attempt is made, the RSEXEC simply informs the user and take no further responsibility for updating that particular file copy.

A completely automatic multi-image file facility would be non-trivial to implement. It would require persistence in updating the file copies. In addition, it would require solution of difficult concurrency control problems that would occur should several users attempt to update the same multi-image file simultaneously. While solutions to such persistence and concurrency control problems are important, they were felt to be beyond the scope of the RSEXEC effort. Recently several researchers have made encouraging progress on the concurrency control problem [34].

RSEXEC supports a feature called device binding. A user can instruct RSEXEC to interpret a particular device name as referring to a particular device at the host he specifies. After a device name has been bound to a host in this way, use of the name without host qualification is interpreted as meaning the named device at the specified host. The binding of devices can be changed dynamically during an RSEXEC session. In the context of the previous example, the sequence of commands:


```
-BIND (device) LPT (to site) BBN-TENEXB  
-LIST (file) DATA.NEW  
-BIND (device) LPT (to site) USC-ISI  
-LIST (file) DATA.NEW
```

produces two listings of the file DATA.NEW; one is printed by the line printer (device LPT) at BBN-TENEXB, the other by the printer at USC-ISI.

An execution environment is implemented for programs. There is a RUN command which can be used to place a program into execution. The RUN command is currently implemented to run the program specified on the local host (i.e., the host on which the RSEXEC command interpreter runs). The program to be run is retrieved from the RSEXEC file system and may be a standard subsystem, such as a text editor or language processor, or a program written by the user. In dealing with a remote file as a parameter of the RUN command, the RSEXEC retrieves a copy of it which it places into execution under its direct control

A program under RSEXEC control executes in the environment of the RSEXEC file system. File operations are interpreted in the context of the the entire network rather than the limited context of the local host operating system. Thus, a program can reference non-local files using either complete or partial pathnames in the same ways an RSEXEC user does. Partial pathnames are interpreted with respect to the user's composite directory. As a result a program can access both remote and local files in a location independent manner, and need not be

rewritten to execute in the network environment. The file movements necessary to complete program file references are performed by the RSEXEC.

To allow users to run programs on non-local hosts RSEXEC implements a command similar to the ATF START-JOB command discussed in Section 4.1. Multiple simultaneous remote jobs are permitted. Like ATF systems, RSEXEC obtains information necessary to create jobs on remote hosts from the user's profile. Programs started on remote hosts via the RSEXEC's ATF function run in the context of the remote host operating system rather than in the RSEXEC program execution environment. While the underlying RSEXEC system structure (to be described in the following section) can support the RSEXEC execution environment for remotely executing programs, the current implementation does not do so.

Another important class of RSEXEC features provide various types of system status information and support interactions among users. There are commands which allow users to obtain information such as the current loads of the constituent host systems (e.g., the number of active users, load factors) and the names of users currently logged into the various systems. In addition, a user can request that his terminal be directly "linked" to the terminal of another user (who may be logged into either the local system or a remote system) in order to engage in an on-line dialogue.

One of the ideas investigated in the context of the RSEXEC system was that of providing expanded functionality for small machines by obtaining services from larger ones [17]. Initial experimentation involved the status query and user interaction functions mentioned above. Because these functions were felt to be generally useful, they were made accessible to all network users without requiring them to first log into one of the RSEXEC host systems. This has been particularly beneficial to the many users who access the ARPANET via TIPS [72], mini-hosts designed to provide terminal access to the network for users who are remote from the hosts they normally use. The TIP itself provides no computational or user services beyond the ability to "connect" a terminal to a remote host. Thus, it does not directly provide sophisticated information and user interaction functions. However, the TIP provides a command that connects to an RSEXEC capable of providing these services.

Because of the success of these initial resource sharing experiments, this approach of using resources of the large hosts to support small host functions was used to implement access control and accounting for TIPS. Due to the absence of long term storage, a TIP is incapable of authenticating its users or maintaining long term usage information. However with some assistance from RSEXEC hosts it was possible to implement TIP access control and accounting. When a user activated a TIP terminal port, the TIP would automatically connect to an RSEXEC for user authentication or login. Should the user fail to login

successfully, further use of the TIP would be denied. After a successful login the user would be permitted to use the TIP in the normal way. As part of the login procedure, the RSEEXEC would pass the user's unique network ID to the TIP for accounting purposes. The TIP would then begin to account for the user's "connect time" and network message traffic. Periodically, at the end of the user's session, and whenever the TIP's dynamic accounting counters were about to overflow, the TIP would send the usage data along with the user's ID to an RSEEXEC host for storage and subsequent processing and billing (1)

4.2.3 RSEEXEC Implementation Approach

The distributed agent implementation approach is used for RSEEXEC (See Figure 8). For each active RSEEXEC user there is an instance of the RSEEXEC program which acts as a command interpreter and program execution monitor. On each host there is RSEEXEC service process called RSSER. Its principal task is to make the resources of its host accessible to remote users. Interactions between RSEEXEC and RSSER processes are governed by a set of conventions called the RSEEXEC/RSSER protocol.

The method of integrating a new host type into the system is to implement RSEEXEC and RSSER programs for it. The functional

-
1. This access control and accounting mechanism has been disabled for a number of non-technical reasons. The mechanism is discussed here to illustrate the potential of the approach.

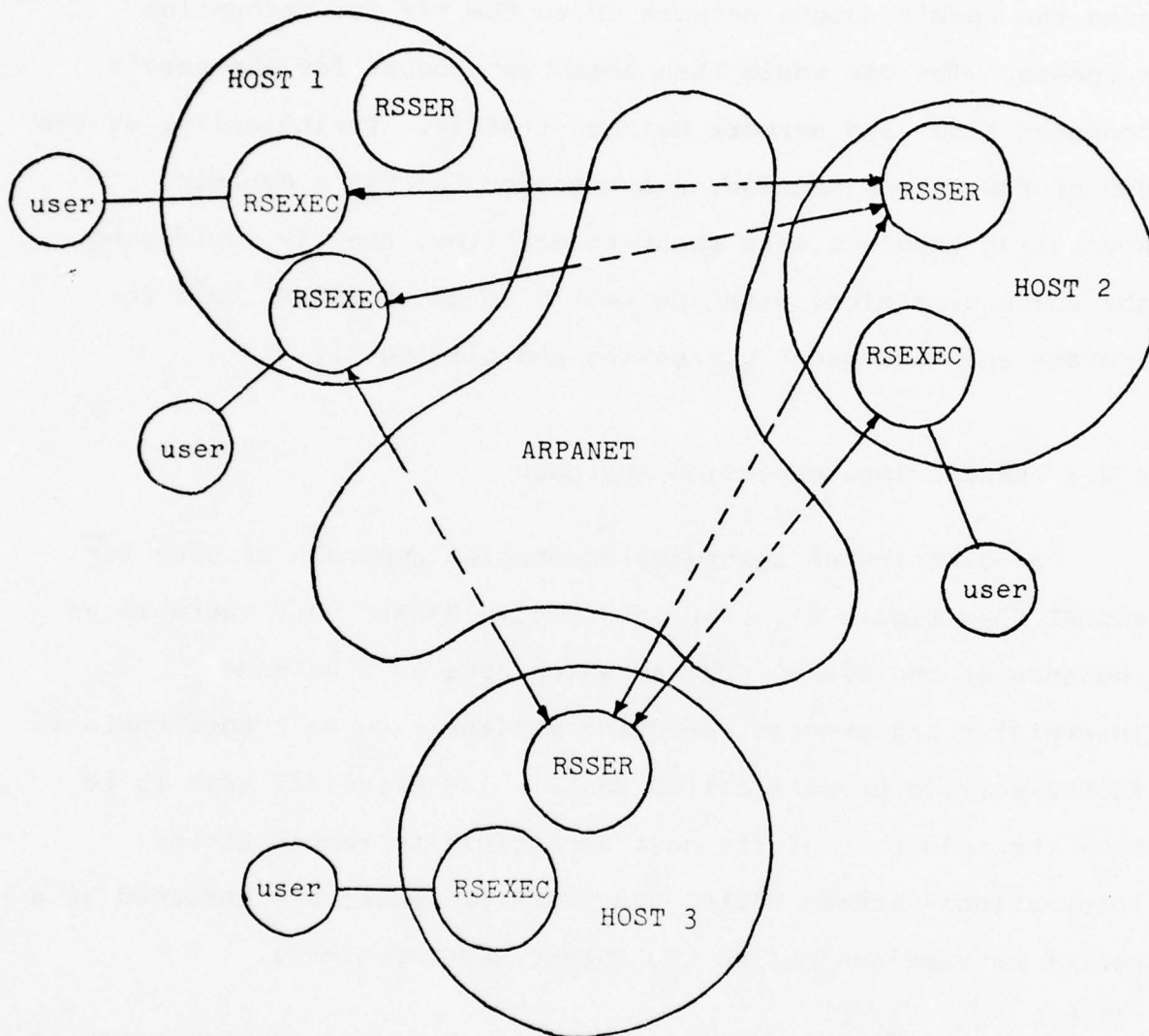


Figure 8.

The distributed agent approach is used for the RSEXEC implementation. The agents are RSEXEC (user agents) and RSSER (host agents) processes.

requirements of the RSSER program are specified by the RSEEXEC/RSSER protocol. The philosophy for an RSEEXEC program for a given host type is that it should provide local users and programs access to the combined resources of the RSEEXEC hosts in the same style that the local operating system provides access to local resources. Thus, a user's interactions with a TENEX RSEEXEC program would be similar to his interactions with a single host TENEX system, while the interactions with a Multics RSEEXEC program would be similar to interactions with a single host Multics system. Although the style of interactions can be expected to vary from host type to host type, the functionality supported by RSEEXEC programs for different host types would be similar.

The RSEEXEC/RSSER protocol is a process-to-process protocol designed to support the interprocess communication required to implement the RSEEXEC functions. It includes commands to query the status of users at a host, to manipulate and move files, to establish terminal links, to start programs running, etc. It was designed to be independent of host type. The fact that non-TENEX hosts have found the protocol relatively easy to implement suggests that the design was successful in that regard. The protocol is machine oriented in the sense that its syntax and control structure is concise and well-defined. Thus, the RSEEXEC does not face the problem of formulating requests and parsing responses intended for human users that ATF systems face.

The protocol itself is connection oriented. It uses communication paths or connections between processes in two ways. Command connections are used for the exchange of protocol commands and responses between RSEXEC and RSSER processes. In addition, there are auxiliary connections, established at the instruction of an RSEXEC process, between RSEXEC and RSSER processes or between two RSSER processes. Auxiliary connections are used to support file movement and interactive terminal traffic.

A large part of what the RSEXEC program does is to locate the resources necessary to satisfy user requests. Some requests can be satisfied directly whereas others may require interaction with one or more remote RSSER processes. For example, an APPEND command may involve interaction with none, one or two server processes depending upon the location of the files. Consider the APPEND example above and recall that the files DATA.NEW and DATA.OLD are at USC-ISI and BBN-TENEXB respectively. Further, assume that the APPEND request is initiated at an RSEXEC running at SRI-KL. After determining the file locations from the user's composite directory, the RSEXEC connects to the servers at USC-ISI and BBN-TENEXB. Next, using the appropriate protocol command it instructs each to establish an auxiliary connection to the other. Finally, it instructs the server at USC-ISI to transmit the file DATA.NEW over the auxiliary connection, and the server at BBN-TENEXB to append the data it reads from the auxiliary connection to the file DATA.OLD.

An issue basic to the RSEXEC implementation concerns handling the information necessary to access files. In particular, how much information about non-local files should be maintained locally by the RSEXEC? The advantage of maintaining information locally is that requests requiring it can be satisfied without incurring the overhead of first locating the information and then accessing it through the network. Certain highly interactive activity would be precluded if it required significant interaction with remote RSSER programs. For example, file name recognition and completion would be unusable if it required direct interaction with several remote server processes. Of course, it is impractical to maintain information locally about all files at all network hosts.

The approach taken by RSEXEC is to maintain information about the non-local files a user is likely to reference, and to acquire information about others from remote RSSER processes as necessary. It implements this strategy by distinguishing internally three file types:

- files in the composite directory.
- files resident at the local host which are not in the composite directory.
- all other files.

Information about files of the first type is maintained locally by RSEXEC. It can acquire information about the second type directly from the local operating system. No information about the third type of file is maintained locally; whenever it is

needed, it is acquired from the appropriate remote RSSER. File name recognition and completion and the use of partial pathnames is restricted to files of the first two types.

The composite directory contains an entry for each file in each of the component directories currently ACQUIRED. At the start of each session the RSEXEC constructs the user's composite directory by acquiring information from the RSSER programs at the hosts specified in the user profile. Throughout the session it modifies the composite directory, adding and deleting files and directories, as necessary. The composite directory contains frequently accessed information (e.g., host location, size, date of last access, etc.) about the user's files. It represents a source of information that can be used without incurring the overhead of obtaining it from a remote host each time it is needed.

It is important to point out that the contents of the composite directory are maintained only for the duration of an RSEXEC session. The composite directory is not maintained as such between RSEXEC sessions. Long term maintenance of information about RSEXEC files is accomplished directly by the file systems of the constituent host operating systems.

The RSEXEC file system is implemented directly by the file systems of the constituent host operating systems. Because this can lead to some anomalous situations, RSEXEC regards the composite directory as an approximation (which is usually

accurate) to the state of the user's files. The state of a given file is understood to be maintained by the operating system at the site where the file resides. The RSEXEC is aware that the outcome of any action it initiates involving a remote file depends upon the file's state as determined by the appropriate remote operating system, and that the state information in the composite directory may be "out of phase" with the actual state. It is prepared to handle the occasional failure of actions it initiates based on inaccurate information in the composite directory by giving the user an appropriate error message and updating the composite directory. Depending upon the severity of the situation it may choose to change a single entry in the composite directory, reacquire all the information for a component directory, or rebuild the entire composite directory.

As with the ATF systems discussed earlier, the user profile is critical to the operation of the system. Profile information is entered by the user through the RSEXEC profile editor. The profile editor verifies information supplied by a user before entering it into his profile. It does this by checking the validity of password and account information with an appropriate RSSER process. As with all interactions between RSEXEC and RSSER processes, this validity check is governed by the RSEXEC/RSSER protocol.

The user profile is stored in a file and read in by RSEXEC as part of its initialization procedure. Sensitive profile

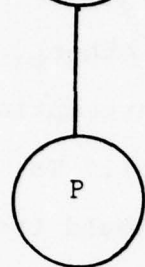
information, such as user passwords, is protected by encryption. When the user invokes RSEXEC he must correctly supply as a password a previously selected encryption key. RSEXEC uses this key to decrypt sensitive information into a useable form.

To ensure highly available RSEXEC service, the profile is implemented as a multi-image file. An image of the user's profile is maintained at every component directory specified in it. This ensures that RSEXEC services will be available to a user when some of the hosts specified in his profile are inaccessible, so long as he can access one of the hosts.

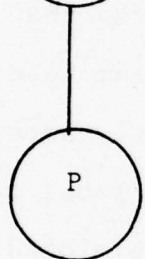
The RSEXEC program execution environment is implemented by a technique called "encapsulation" [16]. The RSEXEC intercepts certain file system operating system calls made by programs before they reach the local operating system. It interprets these file operations in the context of the RSEXEC network file system. Operations that reference local files are completed by passing them directly to the local operating system. Remote operations are forwarded to the appropriate remote RSSER server program (See Figure 9).

When a program opens a file that is remotely stored the RSEXEC instructs the remote RSSER process to open the file. The file is kept open at the remote site until the program closes it, or until communication between the RSEXEC and RSSER is interrupted. By keeping the file open at the remote site, mutual exclusion for multiple readers and writers can be implemented

NOS Study



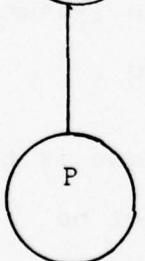
- . P initiates file operation
- . Operation trapped
- . P suspended
- . Control passed to RSEXEC



NETWORK

RSSER

File



- . RSEXEC and remote service RSSER complete operation
- . P resumed

Figure 9.

RSEXEC uses a monitor call intercept mechanism to support uniform access by a user program (process P) to local and remote files. Access to remote files is accomplished by interacting with a remote RSSER process.

using the file system mechanisms of the constituent host operating systems. If the file is opened for reading, other readers can open it. If it is opened for writing, no other writers or readers can open it (unless all processes attempting to access it explicitly requested non-exclusive access). To prevent an open file from being permanently locked, should the remote process that opened the file or the host it runs on crash, the RSSER process will close a remotely opened file if the communication path with the remote RSEXEC unexpectedly breaks.

When a file is opened for reading, RSEXEC retrieves the entire file and makes a local copy of it before allowing read access to continue. Subsequent read operations directly access the file copy. When a file is opened for writing, a local copy is created to hold the written data. When the file is closed the written data is transmitted to the remote site. To avoid unnecessary file movement RSEXEC maintains a cache of files it has had to move. A reference to a cached file is satisfied through the cache rather than by moving the file again (assuming, of course, the file has not been modified since cached).

An important benefit of the encapsulation technique is that programs written for a single host environment need not be rewritten to operate within the network environment. Thus the value of existing software; such as text editors and language processors, is significantly enhanced since the software execution environment now includes remote as well as local data files.

As noted above, the program execution environment is supported only for locally executing programs. It would be relatively easy to extend the RSEXEC system to support this environment for remotely executing programs. The RSEXEC/RSSER protocol includes commands an RSEXEC can use to start a program running under a remote RSSER process. At present because the RSSER processes do not use the encapsulation technique, these programs execute in the environment of the remote operating system. By transmitting the contents of a user's composite directory to the RSSER that controls the remote program, the RSEXEC program could supply the information required by the RSSER to perform the encapsulation necessary to implement the RSEXEC program execution environment.

To support local RSEXEC users, each RSSER program maintains information on the status of the remote RSSER programs. It does this by periodically exchanging status information with the other RSSER processes. The status information it maintains for each host indicates whether the RSSER process at the host is up and running, the current system load at the host, and so forth. When the RSSER process at a host first comes up, the status table is initiated by marking the other hosts as down. After a particular host is marked as down, the RSSER must collect a number of status reports from it before it can mark the RSSER at the host as up and useful.

Whenever an RSEXEC program needs service from some remote RSSER, it checks the status information maintained by the local server. If the remote RSSER is indicated as up, it goes ahead and requests the service; otherwise, it does not bother. Use of this status information makes it possible for the RSEXEC to make a quick decision regarding the ability of a remote site to provide it services. As a result, a user does not experience a long delay while it tries to establish communication with a non-operable remote RSSER. The disadvantage of this approach is that, due to random fluctuations in network or host responses or the failure of the local RSSER, occasionally a remote RSSER capable of supporting RSEXEC service is considered down by a local RSSER.

The RSEXEC functions provided to the TIPs (and other non-TENEX hosts) are supported by a pool of hosts. When the TIP needs an RSEXEC, either to authenticate a user and initialize accounting procedures or because a user has explicitly requested one, it selects a host from the pool. It does this by broadcasting requests for the service to a number of hosts in the pool and selecting the first that responds. This simple selection mechanism enables the the service load to be distributed among the hosts in the pool, and ensures that the service is available as long as at least one of the hosts is accessible (1).

1. An RSEXEC accessed in this way does not provide the full

4.2.4 RSEEXEC as an NOS

The RSEEXEC system provides a number of improvements over ATF systems. Perhaps the most significant one is that users have a more unified view of network services. After a user has supplied the information necessary for RSEEXEC to configure a computing environment for him, it is possible for him to access resources without regard to the network or the boundaries between hosts. This information is maintained in his user profile and need be specified only once, although the user may wish to dynamically control his computing environment by specifying parts of the information to be used by RSEEXEC and parts to be ignored. After configuring his computing environment, a user need not, for the most part, be concerned with the location of files and programs. Because the RSEEXEC system assumes responsibility for locating the resources required to satisfy requests, he can issue many commands in a host and location independent manner.

The RSEEXEC program execution environment is an important feature though it has limitations. It supports one step, location independent data access. Because data movement is implicit, it is unnecessary for a user or his programs to explicitly move data files to the accessing host. When an executing program attempts to access a file, RSEEXEC moves the file if it is not already locally stored. This is a significant

repertoire of RSEEXEC services to a user since the user has not identified himself by a login.

improvement over ATF systems for which data access is a two step operation.

The RSEXEC exhibits several deficiencies which for some users may limit its value as an NOS. These include the following:

- It is primarily a homogeneous system with only partial integration of heterogeneous hosts.

TENEX and TOPS-20 hosts are totally integrated into RSEXEC. Other host types are partially integrated in that they support the RSEXEC status query and user interaction features but only limited file system activity. Data translation problems are avoided by allowing the movement of only text files between TENEX/TOPS-20 hosts and other host types. No attempt is made to resolve the incompatibilities between the file name syntax conventions used by the different host types. To do so would be difficult given the direct way RSEXEC makes use of the constituent host file hierarchies, and its failure to maintain long term resource management information. The NSW and ELAN systems, discussed in the following sections, handle this problem by introducing their own file name syntax, by maintaining their own file directories for pathname interpretation, and by using the file systems of the constituent hosts only to store NOS files.

- User process capabilities are limited.

The user interface supported by RSEXEC is quite powerful. However, the execution environment supported for user processes is available only to processes that execute locally. It could, without much difficulty, be extended for remotely executing processes as discussed above. A more significant deficiency is the absence of any user level interprocess communication facility. As a result, distributed multi-process application programs must use the host network control programs (NCP) directly to communicate.

- The system is visibly distributed.

Although a user who has set up his profile need not, for the most part, concern himself with host boundaries, the host boundaries and the network are not totally transparent. Complete file pathnames include a host name field; component directories contain a host name field; and, a user must establish accounts at each of the different hosts he includes in his user profile. Most effective use of RSEXEC requires, from time to time, the use of commands which explicitly deal with component directories and hosts.

- Interference between NOS and non-NOS activity is possible.

The resources RSEXEC manages are not solely under RSEXEC control. Consequently, RSEXEC resources are not protected from non-NOS access. For example, the same user file can be

manipulated both from within RSEEXEC and directly through the host operating system where the file is stored. This is a consequence of the RSEEXEC implementation approach. The RSEEXEC NOS environment is built directly upon the constituent hosts and only minimal resource management tables are maintained by RSEEXEC. As a result an RSEEXEC user may not be aware that a resource, such as one of his files, is being manipulated outside of the RSEEXEC NOS environment. The impact on the user is that his subsequent attempts to manipulate the resource may fail. The RSEEXEC handles these failures by informing the user and letting him take any corrective action himself.

The NOS systems described in the following sections each address some or all of these deficiencies. They do so in different ways and with different degrees of success. There is some question whether visibility of distribution is a deficiency; this is discussed further in Section 5.

As is the case with ATF systems, the importance of these limitations to a particular user depends upon the user's needs and his level of sophistication. RSEEXEC provides all of the fundamental ATF functions described in Section 4.1 as well as some considerably more sophisticated ones. Consequently, effective use of network resources through RSEEXEC, in general, requires less sophistication than similar use of them through an ATF system would require.

4.3 The National Software Works System

The National Software Works [18] is a network operating system being developed to provide an environment to support software production. The system is designed to provide programmers uniform, convenient access to a wide variety of software production aids, called tools, and to provide managers of programming projects access to a collection of management tools for monitoring and controlling project activities. Software production aids include conventional tools such as text editors, simulators, compilers, interactive debuggers, emulators, and test data generators, as well as experimental aids being developed as part of various research projects, such as program verification systems and systems supporting program development methodologies.

Although tools such as these, which span the software development process from design through implementation and checkout, have been available on a variety of computers for many years, they are seldom applied together in an effective way to support software implementation projects. One reason is that existing tools have been implemented for different computer systems, and programmers typically do not have access to the range of machines that house them. Furthermore, even if they did, programmers would have to master a variety of different host operating systems and command languages, and deal individually

with basic interhost incompatibilities to use the tools. The NSW system addresses both of these problems. It is designed to support access to a wide variety of tool bearing hosts (TBHs) and to provide a simple, uniform means to invoke a tool regardless of the host that provides the tool. One of its goals is to mask as much as possible the incompatibilities of the heterogeneous hosts that support its tools.

4.3.1 Scenario of NSW Use

The initial release of the NSW system included as TBHs PDP-10 hosts operating under TENEX, Honeywell 6000 hosts operating under Multics, and an IBM 360/91 operating under OS. Part of the user community is a group developing software for the AN/UYK-20 computer, a small computer that does not itself support a wide range of software development aids and is generally configured for production use rather than program development. The NSW support of the edit-compile-debug cycle for this group illustrates the intended use of the system. Interactive text editors that run on TENEX and Multics are available for program preparation, modification and documentation. Source programs are compiled by language processors that run on the 360/91 and loaded to form executable AN/UYK-20 object modules by loaders that can run either on the 360/91 or on TENEX. Interactive debugging for the AN/UYK-20 programs is done within NSW using an AN/UYK-20 debugger tool which runs on a TENEX TBH. This interactive

debugger is, in fact, a multi-computer tool. Part of it runs on TENEX and part of it runs on a microprogrammable computer, the MLP-900, which is connected as a peripheral to one of the TENEX TBHs. The MLP-900 has been programmed to emulate the AN/UYK-20 and operates under the control of an interactive controller/debugger module that executes as a TENEX NSW tool. After the several edit-compile-debug cycles required to produce a debugged AN/UYK-20 program, the user can "export" his debugged AN/UYK-20 load modules from the NSW for final checkout and operation on a real AN/UYK-20 machine.

To use any tool, a user simply specifies the tool by name (e.g., CMS2M, the AN/UYK-20 compiler) and the NSW system starts an instance of the tool and connects the user to it. The user need not know which TBH supports the tool nor the command language of the particular TBH. He need only learn the NSW command language. To support tool execution the NSW implements a distributed file system. All NSW tools utilize this single, NSW-wide file system for their file references. When a tool attempts to access a data file (e.g., a source program for a compiler) the NSW system ensures that the access references the correct file independent of its actual location. If the referenced file is stored on another host system, the file is automatically transported to the referencing TBH. In addition, NSW performs the necessary file transformations to deal with

basic interhost file system incompatibilities. Thus, the user and the tools he employs need not concern themselves with the location of data files, the details of the file systems on the various constituent TBHs, the movement of files between hosts, or the data translations that may be required to make a file created on one host type usable on another host type. At present, the file translations provided are limited to those required to support the existing set of NSW tools. The general problem of data translation is a difficult one which has not yet been fully addressed in the NSW system.

It should be emphasized that the primary objective of the NSW project is not tool development. Rather it is the development of a system framework to support uniform access to and integrated use of a diverse collection of existing and to-be-developed tools that execute on a wide range of TBHs under a variety of different operating systems.

4.3.2 NSW Design Decisions

The NSW is an example of the meta-system approach to developing an NOS. Both the heterogeneity of the constituent hosts and the direct use of the existing host operating systems as building blocks are direct results of recognizing the prior existence of important software development tools on a variety of hosts. Although the direct use of the constituent host operating

systems for supporting the execution of existing tools was an important design consideration, so too was the goal of masking the user's direct use of the host systems in an effort to achieve interhost compatibility and network transparency. NSW supports what we have termed turn-key usage. That is, the user is generally limited to invoking prepackaged programs, NSW tools, to meet his computational requirements. General application programming is not supported in the current NSW; this point is discussed more fully later in this chapter. However, a limited application programming environment is supported for a special class of users, tool builders.

Two key decisions served to establish an NOS framework for the architectural development of NSW. First, to achieve a transparent network oriented system and to be able to incorporate and experiment with various project management strategies, all resource allocation decisions are to be made by the NSW system itself without user assistance. Second, there is to be an NSW administrative organization which will relieve the user from dealing with each constituent host to obtain access rights and accounts. NSW supports the concept of a central system administration which serves as the broker for all resources available through NSW. Currently, NSW purchases from various TBH systems the resources to support its users, and has the capability of redistributing the individual tool charges through a centralized NSW accounting system.

These global NSW concepts were followed by more concrete design decisions as the NSW was transformed from a concept to a system design. Perhaps the most important of these early design decisions, in terms of NSW as an NOS model, was to define a single functional module to handle the NSW functions for resource allocation and access control and to maintain the file system catalogue. Centralizing these functions in a component which became known as the Works Manager was a means of satisfying some of the goals of uniformity among heterogeneous hosts with a simple design which has a straightforward implementation. In general, whenever an NSW resource is needed, the Works Manager is invoked. Of course, resource allocation decisions of various types are made at all levels of the NSW, and the Works Manager is not included in all of them. However, NSW access to the major visible NOS resources (tools and files) is coordinated through the Works Manager, which maintains the user, tool and file data bases necessary to handle these activities.

The location of the file catalogue exclusively within the Works Manager means that handling a tool's file references requires Works Manager participation. Rather than requiring that all file references be resolved in a batch-like fashion before running the tool, it was decided to support truly interactive tools. This established the need to provide an NSW execution environment to communicate with the Works Manager whenever a tool dynamically accesses an NSW file. After the need for it was

established, the NSW tool execution environment has expanded to include other parts of the tool interface. Other important design decisions evolved from the original conception of NSW. These include the need to incorporate automatic mechanisms for file migration and translation to accommodate an integrated, multi-host tool set, and the need to establish a uniform NSW user interface which exists apart from any of the interfaces of the participating hosts. Both of these decisions result from the attempt to diminish the incompatibilities of heterogeneous hosts as seen from the perspective of the NSW user.

4.3.3 NSW System Structure

Based on these goals and preliminary design decisions, the NSW system architecture took shape. We will now describe the resulting system structure in sufficient detail to highlight its characteristics as an NOS model.

The NSW design is based on decomposing the system into individual components which are responsible for selected parts of the NSW functionality. There are separate modules designated for functions such as the user interface, file motion, tool control and system communication support. It is useful to think of the principal components of the NSW system as processes which cooperate to provide NSW services. These components include front end (FE), works manager (WM), TBH foreman (FM), and file

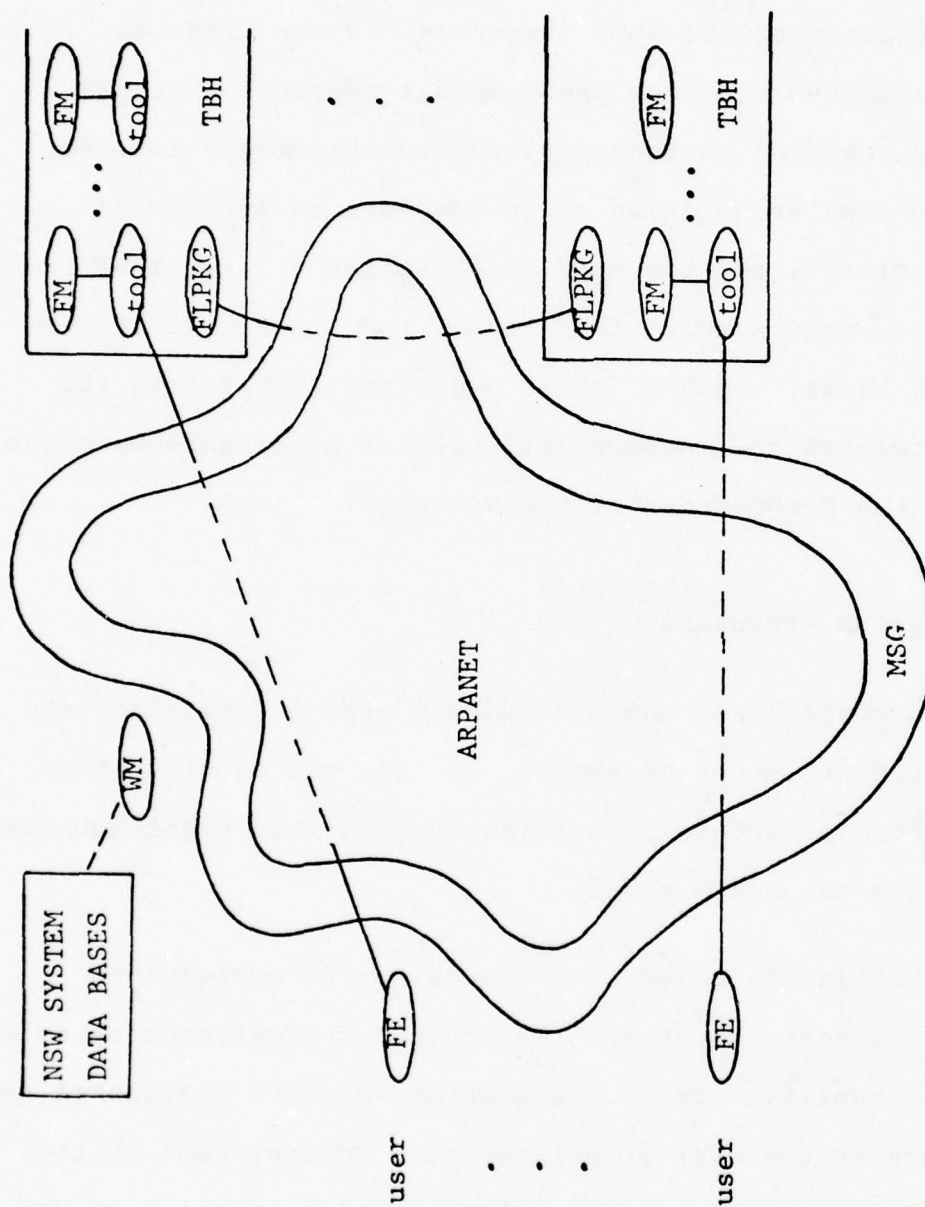


Figure 10.
Relation of NSW components including front end (FE), works manager (WM), foreman (FM), and file package (FLPKG) packages.

package (FLPKG) processes (see Figure 10). Each active user has a dedicated FE process which acts as his interface to the NSW system. The FE acts principally as a command language interpreter making requests upon other components as necessary to satisfy user commands. The FE process supports a standard NSW user interface, including a standard set of control functions and commands, regardless of the host on which it is actually implemented.

As mentioned previously, the WM is the resource allocation and access control module for the NSW system. All attempts to access NSW resources, such as tools or files, must be authorized by the WM. To perform its task, the WM maintains data bases such as an NSW file system catalogue, tool descriptor information, and user authentication information. It also maintains lists of the rights and privileges of each user known to the system. Interactions between WM processes and other system components occur on a transaction oriented basis. That is, the system does not dedicate a single WM process to each active user for the duration of the user session. Rather, WM processes are dynamically allocated (and deallocated) as necessary to support a user session. For example, when a user initiates a command that requires access to an NSW resource, a WM process is allocated to handle requests related to that command. Upon completion of the command the WM process is deallocated (i.e., either returned to a

pool of free WM processes or terminated). Continuity across such instances of WM service is achieved through the use of a shared dynamic data base which depicts the momentary state of NSW, including lists of currently logged in users and their active tools.

The tool bearing host FM [19] is the tool's interface to the NSW. When a user requests the start of a tool, an FM process on the appropriate TBH is allocated for the duration of the tool session. The FM process provides the NSW execution environment for the tool and controls its operation. This execution environment differs somewhat from the standard environment provided to the tool by the local TBH operating system. For example, when a "file open" operation is initiated by a tool, the operation must be processed in the context of the entire NSW rather than that of the local host operating system (see Figure 11). The FM process responds to such an attempt by interacting with a WM process to complete the file reference. The WM process consults the NSW file catalogue to verify the existence of the file specified by the FM, and that the user and tool are authorized to access the file. Next, the WM acts to ensure that the file can be physically accessed by the FM/tool. In general, this may require movement of the file to the FM host and possible translation of file data to a form usable by the tool. The FM provides each tool instance with a temporary workspace for file

NOS Study

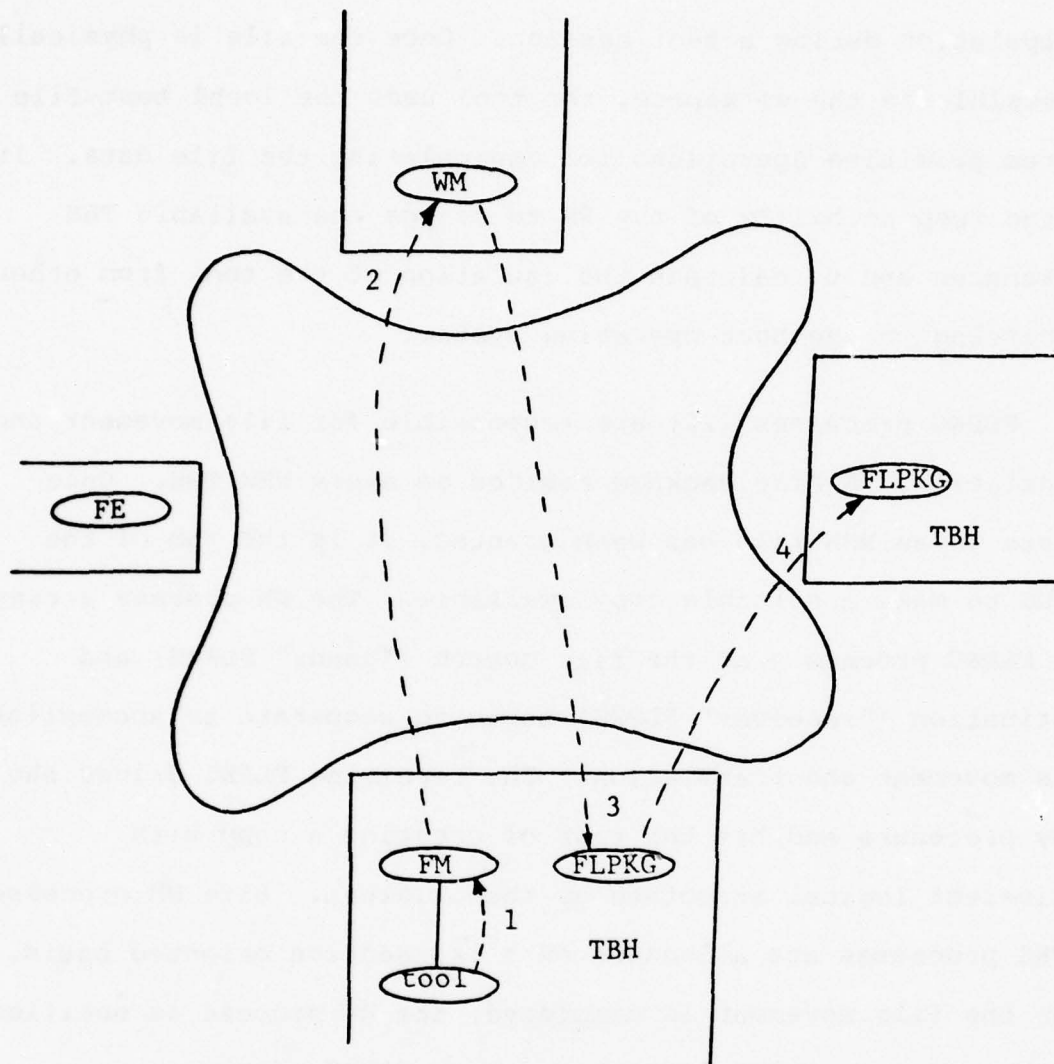


Figure 11.

Tool attempts to access files are fielded by its FM (1). The FM forwards a request to a WM process (2), which instructs the FLPKG at the tool host to obtain a copy of the file (3). The FLPKG does this by interacting with the FLPKG at the site which stores the file (4).

manipulation during a tool session. Once the file is physically accessible in the workspace, the tool uses the local host file system primitive operations for manipulating the file data. It is the responsibility of the FM to manage the available TBH workspaces and to maintain the isolation of the tool from other processing on the host operating system.

FLPKG processes [20] are responsible for file movement and translation. A File Package resides on every NSW TBH. Once access to an NSW file has been granted, it is the job of the FLPKG to make a suitable copy available. The WM process arranges for FLPKG processes at the file source ("donor" FLPKG) and destination ("receiver" FLPKG) hosts to cooperate to accomplish this movement and translation. The receiving FLPKG drives the copy procedure and has the task of creating a copy with equivalent logical structure as the original. Like WM processes, FLPKG processes are allocated on a transaction oriented basis. When the file movement is completed, the WM process is notified by the receiver FLPKG process and both FLPKG processes are deallocated. The WM process then informs the FM process that the file reference can be completed and is itself deallocated. Finally, the FM uses information provided by the WM (e.g., the name of a local temporary copy of the file which the tool may access) to complete the tool file reference.

Communication between the various NSW system processes as well as the allocation and deallocation of those processes is the responsibility of a component called MSG [21]. Every host which is part of NSW has an MSG implementation. It is interesting to note that MSG was designed after the functional component decomposition and modes of interaction for NSW were already established. Although MSG is a general purpose communication facility, it was designed with the patterns of NSW communication in mind, and to specifically satisfy NSW's communication needs. Two addressing modes are supported by MSG. Generic addressing is the means by which a process initiates a transaction with another previously unrelated process. It is used when any process of a given type is acceptable. For example, to initiate the file operation in the scenario above, the FM process sends a generically addressed message to a WM process. Similarly, the FLPKG processes on the file source and destination hosts are activated by generically addressed messages. MSG handles a generically addressed message by allocating a process of the appropriate class (e.g., WM, FLPKG) to receive it. With generic addressing, a process can select a particular host for providing the generic function, or alternatively can let MSG select an appropriate host for that function. The second type of addressing is specific addressing. It is used when the sending process must communicate with a particular process. In the example above, the replies by the FLPKG process to the initiating

WM and by that WM to the initiating FM are sent as specifically addressed messages. When a process executes a receive operation it declares whether a generically or specifically addressed message is requested. In addition to the message mode of communication MSG also supports direct access communication. A pair of MSG processes can request that MSG establish a direct connection between them. These connections are based on the ARPANET host level network connection facility. MSG supports connection requests which are based on the names of the communicating processes, not network sockets as in the standard ARPANET convention. Direct connections are used by NSW processes when extended exchanges are anticipated (e.g. FE to tool) or there is a large volume of data to be transferred (e.g. file transfer).

The NSW is designed to support a wide variety of TBH types. Currently, the system includes three different host types as TBHs. NSW is representative of the distributed agent approach to an NOS. For a host to be integrated into the NSW, it is necessary to implement MSG, FM, and FLPKG components (agents) for it.

4.3.4 NSW File System

The desire for a uniform, logical file system despite heterogeneous TBHs motivated NSW to build its own file resource.

The NSW distributed file system has NSW specific file syntax conventions, file attributes and access control mechanisms. NSW files are NOS meta-resources in that they would not be directly available on the non-NSW host system. In its implementation, the NSW file system is built upon the file systems of the constituent hosts. These file systems are used simply as a storage pool for NSW files. Each storage pool is organized and manipulated by the FLPKG component on that NSW host. The NSW file catalogue is maintained by the WM, and is the link between the NSW-wide attributes of the file and the (potential) multiple copies physically housed in the storage pool. A file catalogue entry includes the NSW name for the file, the location(s) of the file in a constituent host file system, and various other file attributes. Like RSEXEC the NSW system supports the use of partial and complete pathnames for convenience and disambiguation purposes. Unlike RSEXEC, complete NSW file pathnames do not include a host name component.

NSW file operations are logically centralized in that the central WM file catalogue is (almost) always used to resolve file references (1). Additionally, the separation of the catalogue

-
1. TBH FM processes are usually implemented to maintain a "cache" of information about previously referenced NSW files already in the workspace. This cache eliminates the need to interact with the WM when a tool references a file previously referenced within the tool session.

information maintained by the WM from the actual file data maintained by the FLPKG means that these two components must cooperate to make the contents of NSW files available to tools and users. File operations that do not manipulate the file data (e.g. NSW file renaming) require catalogue modifications only, and can be completely handled by the WM alone. The centralized file catalogue approach is somewhat different from that used in RSEXEC. The RSEXEC file system makes more direct use of the constituent host file systems in the sense that it does not maintain a catalogue independent of those of the host operating systems. In RSEXEC, file operations are decentralized. The RSEXEC program acquires file catalogue information with which it maintains the user's composite working directory, by interacting as necessary with various RSSER server programs. This direct use of the constituent host file systems is more feasible in the RSEXEC system because it is designed primarily to work with a network of similar systems. NSW is intended for a heterogeneous configuration and its design goal of isolating itself entirely from any host specific syntax prevents this sort of direct use of the constituent file systems.

An early NSW design decision was to provide tools direct access only to "xerox" copies of NSW files. When a tool opens a file, a copy of the file is moved from the NSW file storage space into the tool workspace maintained by the tool's FM. This

movement occurs even in those cases where the NSW file already resides on the same host as the tool. File access generally requires the cooperation of the FM, WM and one or more FLPKG processes depending on the location of the referenced file. Any necessary file translations are handled by the FLPKG components. The file manipulated by the tool is truly a copy in that any modifications the tool makes to it will not be reflected in the actual NSW file maintained by the WM unless the FM/tool explicitly delivers the copy back into the NSW file system. This "copy on open" mechanism makes it impossible for tools to dynamically share the same file in the way that many modern single host operating systems such as Multics and TENEX permit. Because of this, certain classes of tools, such as data management systems for large data bases, are difficult to support in the current NSW. However, file copying ensures that the NSW always contains an internally consistent version of each file.

To support certain modes of controlled, sharable access to NSW files within the "copies only" discipline, each NSW file has associated with it a so-called semaphore. This semaphore can be set by a tool on behalf of a user in order to warn other potential users that the file may be undergoing change. A "set" semaphore can block attempted access to a file, and thus can be used to coordinate multiple, concurrent file update activity. The NSW system itself does not directly implement this

coordination. It only provides semaphores as the means to accomplish it; the tools or users which require coordinated use of files must use the semaphores in a mutually agreed upon way to achieve the coordinated file activity.

There is a substantial cost associated with retrieving files from and delivering files to the NSW file space. The cost includes file copy operations (possibly network copying) and name resolution operations requiring interhost cooperation. Manipulating files within the tool workspace is far less costly, partly because this local storage is viewed as private to the tool session. It would be very inefficient to use the WM controlled NSW file space to handle the temporary file (scratch file) requirements associated with tool execution. Because of this, tools are provided with primitive operations by which they can specifically create "workspace files", and control the flow of workspace copies back into the global NSW file system. All workspace files need not be delivered to the NSW catalogue. A typical tool session will have all of its delivery operations batched at the end of the session when it is better known which files need to be permanently saved.

4.3.5 NSW Program Resources

Two different types of program resources are present in NSW. These are batch oriented tools and interactive tools. Batch tools are tools which are run without an on-line user and which have had all of their file references resolved before execution begins. When invoking an NSW batch tool, the user enters into a dialog with an NSW system component to specify the NSW files to be used in the tool session. Copies of the appropriate files are pre-staged into a workspace for the tool, and the appropriate job control language statements are inserted into the job stream to cause file references by the tool to resolve to file copies in the tool workspace without requiring user intervention. Batch oriented tools most often do not require NSW support while the tool is active.

Interactive tools dynamically access the NSW file system at various points during their execution. Their file reference patterns are generally not known at job setup time. Interactive tools also have a direct communication path to the on-line NSW user to allow him to control the tool session. To support runtime access of NSW resources by interactive tools, the FM provides an NSW program execution environment. There are two different tool interfaces to NSW. An encapsulation interface is provided to permit software packages that were developed to run under a particular TBH operating system to be installed as NSW

tools. This interface is similar in concept to the encapsulation mechanism used in the RSEXEC system. For NSW tool encapsulation, the FM intercepts certain requests made by the tool of the local TBH operating system (mainly file access requests) and automatically transforms them into similar NSW operations. Ideally, the encapsulation interface allows existing software to be installed into NSW without modification. In practice, the success of the encapsulation approach for a given TBH depends upon the functional commonality between the TBH operating system and NSW, and the degree various tools exercise TBH functions not naturally mapped into NSW functions. For a fully encapsulated tool, the programming abstract machine is identical to that of the local host operating system. Actual differences from the NSW supported abstract machine are mediated totally within the FM. Encapsulation has been fairly successful for the TENEX hosts; however, it is not uncommon for the integration of a new TENEX tool to require minor enhancements to the FM encapsulation interface.

The second tool interface provided by the FM is for tools developed specifically for operation within the NSW environment. This direct interface permits tools to explicitly call NSW functions. Examples of the functions provided to tools are primitives for accessing the NSW file system, for manipulating the files in the tool workspace, and for setting up various kinds

of communication paths to the FE. Generally the FM must interact with other NSW system processes to satisfy these tool calls. The design intent is for the direct tool interface provided by FM processes at different TBHs to be functionally equivalent. The manner in which tools invoke NSW system operations may, of course, vary from TBH to TBH. The abstract machine seen by tools of this class is formed from selected parts of the local operating system abstract machine, augmented with NSW specific constructs in those areas which are dependent on the distributed nature of the system.

Although the major emphasis of NSW support for the program execution environment has been in the area of access to the distributed file system, some preliminary design work has been done toward extending the distributed aspects of the system to the tools. Message communication between the tool and the FE component has been recognized as a useful base for partitioning a tool's functionality between a part local to the user (the FE host) and a part on a larger processing facility (the TBH) to obtain improved performance. A complete NSW design for tool access to MSG interprocess communication message passing facilities is anticipated. Additions to the tool execution environment to include functions for invoking other tools are also contemplated. The ability of programs to dynamically call into execution other network programs, and to organize the

control and communication of these clusters in a convenient way are viewed as important steps in providing an environment conducive to the development of distributed tools.

As we mentioned earlier, the NSW does not support general application programming use. NSW users are not free to augment the public (or system) tool set with private programs which execute as tools under NSW (1). The set of available tools is determined by a WM data base describing the tools. Updates to the data base to support additional tools are governed by NSW administrative decisions. This policy is a consequence of the philosophies for project management and configuration control prevailing at the time the system was designed. As a result, NSW is not a general purpose system in this area, although it would not be difficult to modify NSW to support general application programming.

4.3.6 System Reliability

Because of its modular structure, the NSW system architecture is potentially resilient to individual host failures. The inter-component protocols have been designed to make continued system operation possible in the presence of

-
1. They can, however, to a limited degree run and test software which is under development using host specific debugging programs which are available as tools.

failures of the communications network, the constituent hosts, and the system components themselves. For example, these protocols make extensive use of timeouts to control repeating requests for service, accessing alternative sources, and entering into other recovery procedures. In addition, mechanisms have been developed to recover files trapped in tool workspaces due to TBH crashes or other NSW malfunctions. When a crashed TBH is restarted, or in general whenever a tool session cannot be successfully completed, tool workspaces in use at the time are preserved in a way that allows a user to later retrieve selected files. These mechanisms will soon be augmented to include resuming the tool session in the existing workspace environment whenever possible.

At present the WM is the principal weakness of the system from a reliability point of view. Because the WM and its central data bases reside on a single host, the NSW system is vulnerable to failure of the WM host. A multi-host implementation of the WM function is being designed that would permit continued NSW operation in the presence of WM host failures. In addition, it would also allow the system to gracefully expand as the number of users grows by making it possible to distribute the WM load among several hosts. The principal technical problem here results from the fact that the WM data base is logically centralized. To distribute the WM, its

data base must be distributed. This distribution requires use of a synchronization mechanism to insure that the distributed parts of the data base are maintained in a consistent manner. Several synchronization mechanisms of this sort have been developed recently; see [34] for further discussion of the synchronization problem for multiple copy data bases.

The NSW approach to reliable operation has been to add reliability mechanisms to the system after it had been designed. This effort involved analyzing system weaknesses after its fundamental architecture had already been established, and then developing specific remedies to the perceived reliability problems. An important guideline in developing the reliability plan was the desire to have minimal impact on the various existing NSW components.

4.3.7 Access Control and Resource Allocation

Program and file resource requests, as well as NSW "login" sequences, originate with users or their tools and are initially handled by local NSW agents (FE or FM). The nature of the distribution of information that describes the NSW resources and their access controls generally prevents these agents from completing the requests themselves. The responsibility for making the access control decision is placed within the networkwide WM component. Off-loading it from each TBH and FE

host makes it possible to achieve a uniform facility with a single implementation. Each TBH usually has its own internal access control mechanism but for the NSW user this is subordinate to the NSW-wide facility and is generally not visible. The access control mechanisms of the constituent hosts do, however, play an important role internally to NSW by ensuring the integrity of NSW resources resident on the local hosts and in implementing a controlled tool environment.

Unlike many modern systems, NSW maintains separate mechanisms for controlling access to programs and files. Associated with each user is a "user node" which defines the segments of the NSW file hierarchy and the individual tools which the user can access. A user can dynamically add subordinate user nodes and allow them access to selected tools and file system areas which he himself may access. Augmenting either the tool or file capabilities of a user is handled in a strictly hierarchical basis. Because of the structure of the NSW system, addition of new users or changing existing users' access rights only requires modification of a central data base and does not involve any coordination with non-WM system components.

The WM makes NSW resource allocation decisions regarding host selection for tool execution and NSW file storage. The WM data bases are used to map NSW logical resources into the physical resources on the constituent TBH systems that implement

them. These data bases are structured in a way to accommodate the information needed to support the replication of tools and file copies in order to achieve highly reliable service and performance improvements. Currently, NSW only implements rudimentary mechanisms in these resource allocation areas. However, the centralization of the decision making responsibility makes it convenient to experiment with alternative algorithms without affecting other components.

While the WM defines the NSW policy for resource allocation, the distributed agents are responsible for carrying it out. In doing so they are responsible for allocating the physical TBH objects which support the NSW resources. It is the WM that selects a given host to run a tool and it is the MSG implementation for that host which allocates a process to handle the tool request, and the FM on the host which manages the workspace pool needed to support tool execution. The WM controls the NSW file system name space and it is the FLPKG processes on the various hosts which manage the physical storage medium that supports the file system.

4.3.8 NSW as an NOS

As an NOS model, NSW shows advances over the previous models in a number of important areas. Perhaps the most significant are in the concepts surrounding the integration of a diverse

collection of heterogeneous hosts under a single operating system interface. NSW attempts to create a transparent network interface without resorting to a base level implementation which would obsolete the existing software base. NSW has its own uniform command language and user interface. It has its own file system, incorporating NSW-wide file attributes with a uniform filename syntax. These operating system components have been developed specifically to enable the NOS to support a level of system transparency not otherwise achievable with the heterogeneous hosts. Other parts of the NSW system address the problems of uniformity through protocol and translation mechanisms. Tool encapsulation and translation between file types are examples where these techniques have been used to allow the new environment to support existing software.

NSW has also begun to address some of the managerial issues in supporting a distributed meta-system. The NSW includes the concept of a central administrative organization to provide networkwide coordination of such things as initial access to the facility and tool installation policies. NSW also has a dedicated operations staff which views the maintenance of the system in terms of the unified facility rather than in terms of the individual hosts.

Other significant NSW innovations are in the area of distributed system reliability. Because the NSW functionality is

developed out of a rather complex series of interactions between a number of independent system components on multiple hosts, it could be very vulnerable to partial, temporary failures. An important part of the NSW reliability effort has been to recognize the failure prone nature of the system environment. System components are constructed to tolerate certain degrees of malfunction and maintain their internal consistency across failures. Beyond this, the NSW reliability concepts attempt to ensure the user a consistent image of his NSW resources despite component failures, and to provide him with mechanisms for continuing at an appropriate point whenever failed components come on-line again. There has also been an important design effort to enable the WM function to be replicated so that a single failure cannot cripple the entire system. However, this design is not yet supported by an implementation.

NSW has a number of readily observable weaknesses. At present the most prominent of these is the relatively poor system performance. Performance problems at this stage of NSW development are not surprising since performance considerations were not part of the original design and many of the system concepts are being tried either for the first time or for the first time in a real system. Remedial attention may be able to transform the system as currently designed into a facility with adequate performance characteristics without completely overhauling some of the basic concepts.

The performance problems can be separated into two classes: those related to the design of the system and those related to the implementation of that design. As an NOS model the design related performance problems are the more interesting and are emphasized below.

A factor which contributes to the poor performance of NSW is the layering of system upon system. For NSW in particular, and probably for the meta-system approach in general, there are legitimate reasons for minimizing modifications made to existing constituent operating systems to support the NOS. As a result, the NOS software is usually built upon the existing system, often as application programs. This is advantageous from an ease of implementation point of view. However it is difficult for the NOS that results to match stand alone systems in performing similar tasks unless the constituent host system has adequate facilities to support the efficient execution of system-like functions at the application program level. Most systems currently do not have such facilities.

Two important NSW design decisions also directly impact system performance. First, the centralization of access control and resource management functions in the WM component means that, in general, interhost cooperation is required whenever these functions are needed. For interactive users, responsiveness is a very critical performance measure. Interhost operations are

noticeably less responsive than intrahost operations for almost all network and system configurations. Second, the extensive decomposition of the NSW functionality into a number of independent components each with their own private internal resources and data bases has led to a system organization which relies heavily on extensive communication between components. It is not unusual for a single NSW operation to require the cooperation of three or four individual NSW components. In an environment where communication costs or system scheduling overhead are significant, the large number of interactions required to complete an operation can prove to be a serious performance bottleneck. Even when these costs are not significant, the individual process overhead for handling messages can become significant relative to the operations being performed. Although decomposing a system into a number of independent modules is an effective technique for implementing a reliable system, care must be taken in placing functions and data "near" to where they will be needed. Knowing approximate communication costs for a given system environment is a key factor in deciding how much decomposition can be tolerated. From a performance point of view it is important to prevent access paths from crossing too many host boundaries.

Another possible shortcoming of the NSW is that the TBHs are not self-sufficient and cannot function autonomously within NSW.

This is a direct result of off-loading many of the operating system functions from the TBH to the central WM component. The TBH accepts processing requests only from the WM, and hence the TBH system will become idle if no WM is operational. In this sense, the TBH systems are very tightly coupled to the WM host.

In a number of areas NSW does not exhibit the flexibility normally associated with general purpose systems. Examples have already been cited, such as the inability of users to dynamically augment the set of executable programs and the "xerox copy" mechanisms of the file system which make flexible file sharing impossible. While the considerations which led to these design decisions may be appropriate for a software production environment, they do limit the extensibility of the system and constrain the class of supportable application programs.

Finally, there is a relatively high cost associated with adding a new host type to NSW as a TBH. Implementations are required for MSG, FM and FLPKG components to handle communication, program and file resources respectively. In general, there is little difference between a minimal implementation which adheres to the required functionality and a full scale implementation for these components. This is not surprising given NSW's goal of tightly integrating the operation of a diverse set of computer systems. If overall system conformity were not a major NOS design requirement, as it is for

NOS Study

Models: NSW

NSW, it might be possible to support various levels of system integration, and gradually phase the implementation while some NOS functions were already operational.

The ELAN system design, discussed in the next section, approaches system performance and generality and host autonomy in ways which alleviate some of the shortcomings of NSW as an NOS.

4.4 The ELAN System Design

The Efficient Local Access Network system (ELAN) is a preliminary NOS design. It reflects experience with several network operating systems including RSEEXEC and NSW. Although such existing systems have solved many problems associated with providing access to distributed resources, in many ways they lack the functionality and performance of single-host operating systems. The ELAN design incorporates the strengths of several predecessor systems and provides capabilities absent in previous network operating systems. The concepts embodied by ELAN represent a step beyond ATF, RSEEXEC, and NSW toward a general purpose NOS which presents a unified user interface and a complete program execution environment for accessing distributed resources.

The design presented here for ELAN is not a complete system design. Rather, it serves as a vehicle for an in depth look at some NOS design problems and their solutions. Consequently, some parts of the ELAN design are presented in detail whereas other parts are only briefly described.

4.4.1 General Assumptions and Characteristics

Several guidelines were used during the development of the ELAN model. The first principle, from which the model derives its name, is that the ability to access distributed resources should not be at the expense of efficient access to local resources. A computation that confines its references to local resources should perform as efficiently under the control of the ELAN NOS as it would under the local host's stand-alone operating system. This is not the case for a system such as NSW where the Works Manager must be involved in all major resource control decisions, whether or not the resource being accessed resides at a remote host. Since, in general, the Works Manager resides at a different host than the accessing process, local accesses incur significant overhead. Presenting efficient local access as the first guideline emphasizes our concern about the intrinsic delay of communicating between the constituent hosts of the NOS. The desire is to develop an NOS that is, if not always better, at least is as good as existing single-site systems. The benefits of network-wide accessibility should not be at the expense of performance.

The second principle is to permit autonomous operation by a single component host within the NOS. For various reasons,

including the possibility of communication failure and scheduled down time, the constituent hosts of an NOS may not be fully connected at all times. When this occurs, it should be possible for a host to proceed with a computation that limits its references to reachable resources. RSEXEC displays this characteristic in some of its operations while, as noted previously, NSW requires that the Works Manager host always be a part of the communicating subset. In addition, the types of system features should not change when such reduced accessibility occurs. Users and programs should see the same interface regardless of how many hosts are participating in the NOS, even if the local host is the only NOS host accessible.

At present, general solutions to the problem of concurrency control for updating a distributed data base are costly and complex. The third design principle is that the system should not depend on general solutions to the distributed concurrency control problem. While we felt that NOS capabilities requiring such mechanisms should, if possible, be avoided, we also felt it acceptable to employ special purpose updating schemes where patterns of data base activity are limited and well understood. This guideline reflects an underlying assumption about the communication medium between the constituent hosts. We assume that the delays introduced by transferring messages from one

constituent host to another and the delays experienced in process scheduling are long compared to the delay experienced in accessing values in a shared data base on a single-site computer system. The effect of relaxing this constraint (i.e., assuming that the difference in delay between local interactions and remote interactions were small) would be to merely add capabilities to the ELAN design, not to change the basic design approach.

One of the differences between single-site and distributed systems is the potential for using the autonomous nature of the constituent hosts as the basis for robust operation. In a network environment, this means two things: designing the NOS so that it is robust; and providing mechanisms in the NOS to support the robust operation of application programs. A goal of ELAN is to address these issues to the extent possible in a preliminary design.

Finally, while network operating systems are new facilities, operating systems for the constituent hosts exist and new operating systems for single hosts will continue to be developed. It is generally infeasible to require that new operating systems be designed and implemented for a wide variety of hosts to permit their effective operation in an NOS. The cost would be prohibitive. Furthermore, we feel that it is important to

preserve the large investment in application software that exists or will be developed to run under these single-host operating systems. Thus, the fifth principle is that the ELAN NOS should be implementable upon a set of heterogeneous single-host operating systems.

The primary goal of ELAN as a general purpose operating system is to present a uniform, integrated view of resources distributed among many different constituent systems. To achieve this goal the ELAN model provides operations that can be applied regardless of the relative location of the resources. Examples include primitives for accessing files and for controlling processes. In order to allow optimizations that are possible when host boundaries are not transparent, the ELAN design permits certain modes of usage that rely on the specific location of resources. For example, application processes that engage in tightly coupled sharing that can be supported only if they reside on the same machine. Thus the ELAN system provides for the explicit, but optional, specification of resource location.

4.4.2 Typical Applications

Since ELAN is designed as a general purpose operating system, the types of application programs that must be supported vary widely. There are, however, several classes of programs that should receive explicit attention because together they comprise most of the typical programs expected to run under ELAN. ELAN is designed so that these classes of programs run as efficiently as possible while at the same time other, unanticipated classes of applications can also be supported.

The first class of typical applications are those that reference files distributed over several constituent hosts. The primary requirement for such applications is that files be uniformly accessible and that efficient mechanisms exist for transporting data between constituent hosts. In addition, such programs need efficient mechanisms for cataloging files so that they may treat a collection of many files that span several different heterogeneous hosts as logically related entities. The uniform treatment of distributed files must not only be in terms of naming and file operations, but also in terms of file allocation control, accounting and file protection.

The second class of applications is characterized by those that access several different programs or devices which only run

or exist on unique ELAN constituent hosts. The hosts in ELAN are heterogeneous and thus programs cannot necessarily execute on any machine in the system; similarly some of the hosts may have special purpose or location-dependent devices that are not present on other hosts. The primary requirement for this type of operation is effective process control and interprocess communication regardless of the location relationship between processes.

The third class of applications are those that have a natural subdivision into identical small autonomous units which can act apart from the others, but which also need to act in cooperation with others to form a unified system.

Finally, there is a class of applications that requires extremely robust operation for which the autonomous operation of the constituent ELAN hosts would seem beneficial. This class is more difficult to describe because two attributes must be present: the need for reliable operation and the ability to split the applications into enough parts so that significant recovery can occur when one of the parts fails. Very little work has been done on the problems of dividing a computation into autonomous parts and developing applications which utilize multiple parallel processes.

4.4.3 ELAN System Concepts

To facilitate further discussion about interactions between system resources, we introduce the following relationship: Two resources are said to be adjacent if they reside on the same host; similarly, two entities located at different hosts are said to be remote. ELAN supports a set of location transparent operations that may be performed on any resource regardless of its location relative to the initiating process. In addition, if a process is adjacent to the target resource, certain additional operations on it are permitted. For example, in the ELAN file system the ability of a process to access a file in certain modes is influenced by whether the process is adjacent to the file and by its position relative to other processes that may be sharing the file (1). Only adjacent processes can writably share a file. This restriction exists to avoid requiring that ELAN include a general updating mechanism for distributed data.

Because the ELAN model is observably distributed, optimizations can be achieved when all entities of a computation are adjacent. In the NSW system, the location of resources is totally hidden from the user and even from some components that

1. The access control protection mechanism of the file system also contributes to the admissibility of file references.

make up the system. In RSEXEC, specification of the physical location of resources is optional. For example, a user may, if he desires, include physical location information with commands. ELAN provides both views of resources and allows users and programmers the choice of which to use.

In single-host operating systems there generally are identifiable parts of the system that provide distinct services. Examples include modules for controlling the file system, processes and devices. In an NOS some of these control functions must be distributed among the constituent hosts. For example, the process control function for ELAN will require assistance from the constituent hosts that support user processes. In ELAN the distributed agents (system processes) are called kernel processes (KPs). The ELAN KPs represent a division of the tasks required to implement a general purpose operating system. Types of special task KPs include file management (FM-KP), and process control (PC-KP) kernel processes. When an operation must be performed on a remote resource, a KP at the site initiating the operation will, in general, communicate with a corresponding remote KP to effect the operation on the remote resource.

4.4.4 The ELAN File System

The ELAN model includes a file system which is organized hierarchically, much like the Multics file system [58]. For such a file system, files are catalogued in directories which are themselves catalogued in parent directories. Thus the hierarchy is a tree structure where the leaves are files and the points at which the branches split are directories. The pathname of a file is the sequence of branch names traversed from the root to the leaf. To insure uniformity of file naming and access, the ELAN model supports a single network-wide file hierarchy which is implemented using the storage systems of the individual hosts (see Figure 12).

Care must be taken in designing a mechanism that allows uniform accessibility to files stored at arbitrary network sites in order to prevent unacceptable system performance. If the relationship between the pathname for a file and the physical locations of the directories along the path to the file are unconstrained, then directory hierarchies could develop for which the file name look up operation (i.e., file name to file location operation) is very expensive. In the worst case one interhost interaction for every branch in the tree from the root node to the leaf node would be required. The delay due to interhost

NOS Study

ELAN FILE HIERARCHY

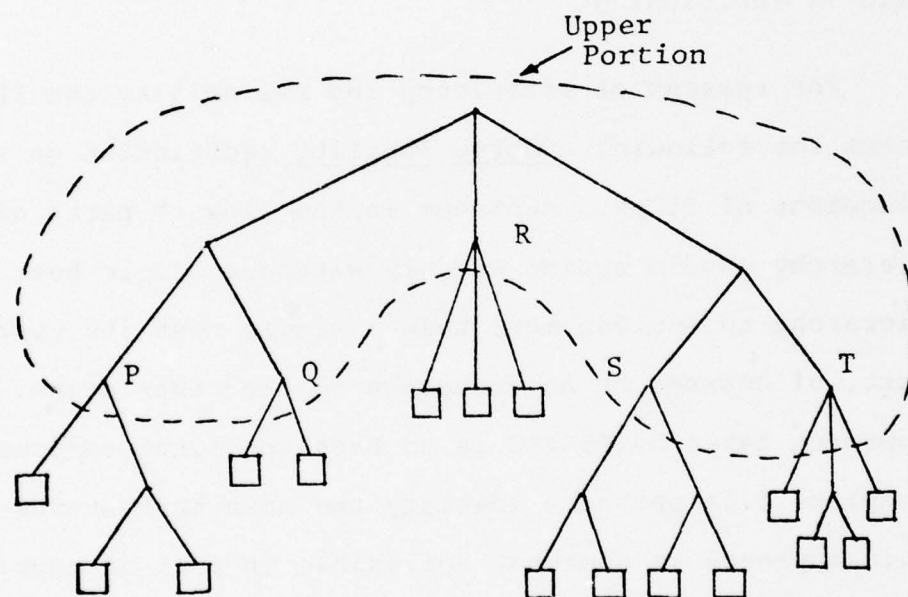


Figure 12.

The representation of the ELAN file hierarchy within which uniform file references are made.

interactions is an issue. In addition, reliability is an issue. If such an unconstrained structure were permitted, then the failure of one node in the collection of constituent hosts could block reference to a file, even though the node that holds the file is functioning.

For reasons of efficiency and reliability the ELAN model makes the following subtree locality restriction on the physical placement of files: subtrees in the "lower" parts of the file hierarchy should reside totally within a single host. For the hierarchy to include more than a single host its upper portion must, of course, be an exception to the restriction. The approach taken by RSEXEC is to have the first component of a complete file pathname identify the host that stores the file. This approach is somewhat inflexible in that it couples the position of a file in the hierarchy with the network topology.

We believe it preferable for the placement and naming of files in the network-wide hierarchy to be dependent only upon relationships that exist among the files. By observing that the upper portion of the file system hierarchy is typically modified infrequently and that the rate at which a part of the hierarchy is changed typically increases with its depth in the hierarchy, it is possible to achieve a workable solution that satisfies the

goal of location independent file naming and reduces the host interactions required to resolve file name references. The top portion of the hierarchy is replicated at every host and updates to it are restricted in a way that can be managed by a simple duplicate data base updating mechanism. The slowly changing upper portion of the hierarchy is used by FM-KPs in file look up operations to translate the initial parts of pathnames into host identifiers (see Figure 13). Thus, although file names contain no explicit location information, the actual host location of a file's directory entry can be determined by a local FM-KP. The user need not remember network configuration details.

There is a common situation that is made inefficient by requiring subtree locality: a file that belongs to a logical set whose directory information is stored at one site but which may only be used at another site. A file that contains executable machine instructions is an example. If subtree locality is strictly enforced, then every time the file is used, it will have to be moved to the site where it executes. To eliminate this problem, the directory information about a file and the contents of the file may be separated by a "link" which points to the site where the file contents reside. This change to the subtree locality rule introduces an additional interhost interaction in some cases but does not affect file referencing efficiency since

NOS Study

DISTRIBUTION OF FILES AMONG CONSTITUENT HOSTS IN ELAN FILE HIERARCHY

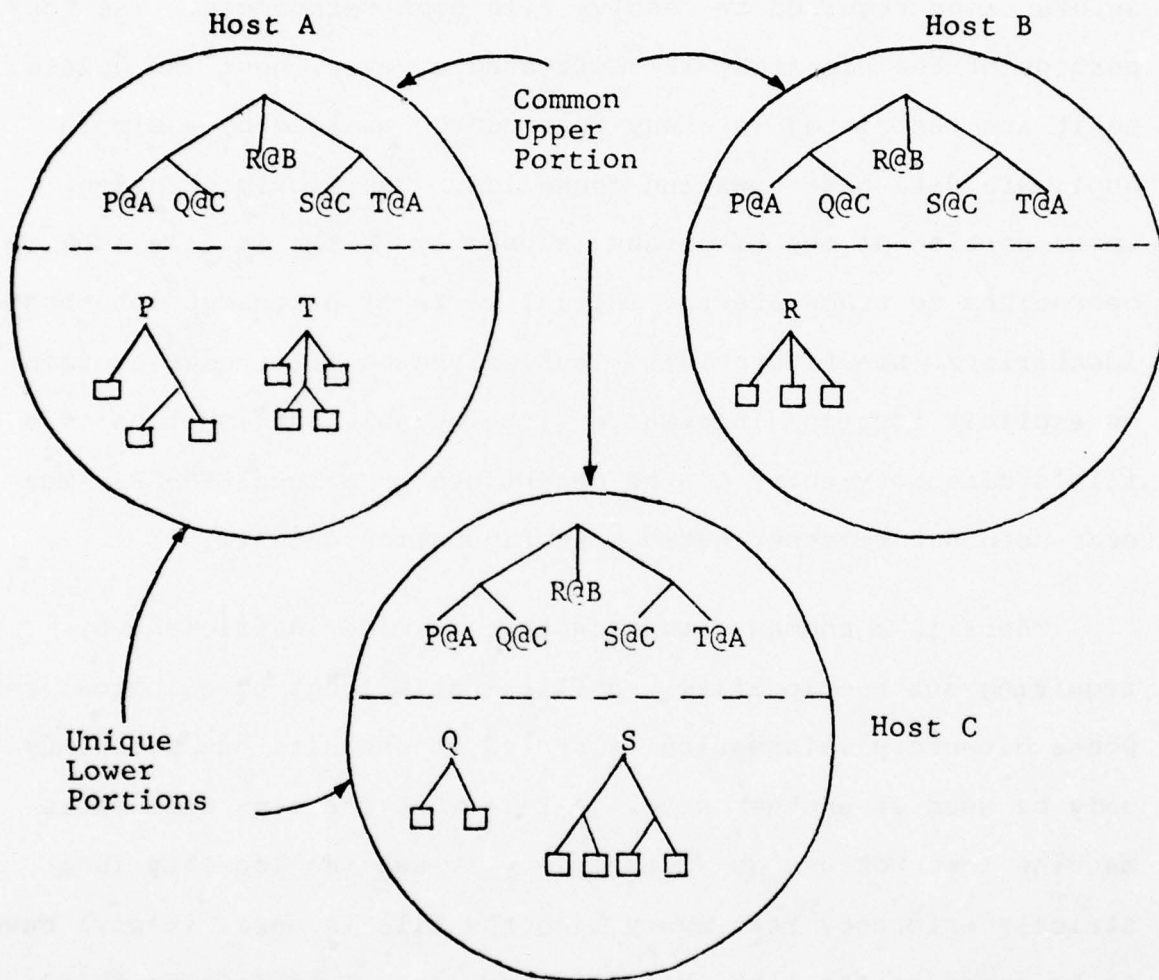


Figure 13.

A possible distribution of the files from Figure 12 among these hosts illustrating the common upper portion and the unique lower portion present at each constituent host.

the file would have had to be transported to the remote host anyway. Thus, two interhost interactions are required for references to remote files which have links to their content, one interhost interaction for remote files without links or local files with links, and no interhost interactions for local files without links.

In RSEXEC the file system directly mirrors the underlying file systems of the constituent hosts. A unified view is possible because of the homogeneity of the constituent host operating systems. The NSW achieves a unified view despite the heterogeneous constituent hosts by building a new file system for which all file cataloging information is maintained by NSW. The ELAN approach is similar to the NSW approach in that the file system is built upon the file systems of the heterogeneous member hosts. Care was taken in both systems so that file operations are supportable by all of the heterogeneous constituent host operating systems.

There are several basic operations that can be performed on files in the ELAN model. In the following descriptions the arguments in brackets ([]) are optional and underlined names indicate result arguments:

```
Create-File(Name, Directory, File-ID, [Site-Location])
Delete-File(Name, Directory)
Open-File(Name, Directory, Mode, Sharing, File-ID)
Close-File(File-ID)
Read(File-ID, Position, Value)
Write(File-ID, Position, Value)
```

Create-File and Delete-File are operations for altering the contents of directories in the file hierarchy. The optional argument Site-Location, if present, signifies that the file directory information should be set up in the indicated directory, but that the actual contents of the file should be stored at the designated site.

Open-File and Close-File are operations that prepare the file system for subsequent reference to the file in question. The mode in which the file will be referenced may take on any combination of the following: Read, Write or Execute. In addition, references to files may be Shared or Exclusive depending on the value of the sharing parameter.

There is also a set of operations for manipulating directories in the file system hierarchies:

```
Create-Directory(Name, Parent-Directory)
Create-Upper-Directory(Initial-Pathname, [Site-Location])
Delete-Directory(Full-Pathname)
Delete-Upper-Directory(Initial-Pathname)
```

The Create-Directory command appends a new directory below the specified parent directory. The location of the new directory is implied by the location of the parent directory because of the subtree locality rule. To add directory structure to the upper part of the tree, a separate operation, Create-Upper-Directory, is provided. Absence of the Site-Location parameter implies the local site. The effect of this operation is to define ordered pairs (initial pathname, site location) which map initial portions of full pathnames into site locations for further name resolution. The data base that stores these ordered pairs is a distributed data base that is replicated at each of the ELAN sites. Once a Create-Upper-Directory operation has occurred, the results of that operation must be visible at every constituent host. The format of this data base (ordered pairs) makes it easy to maintain using a simple technique such as that developed by Johnson and Thomas [40]. The delete operations remove directories or initial pathnames from the file system hierarchy.

4.4.5 The ELAN Process Structure.

Some NOS designs provide uniform but limited access to distributed resources. An example is the way various NOS designs handle user processes. Some provide user processes that either

fill specific roles, as with tool processes in NSW, or are restricted to run on the same host as their parent as in RSEXEC. In the ELAN model, an attempt has been made to provide a more general process capability with fewer restrictions on the way processes can be used.

Processes in the ELAN system are organized in a hierarchical fashion much as they are in the Unix [55] or TENEX [56] operating systems. Figure 14 represents a possible network-wide logical organization of the ELAN process hierarchy. An application process (AP) can create and control the action of inferior processes, which can in turn create and control inferiors of their own. The ELAN system supports these interprocess control functions in a multihost network environment. An AP at a particular host may create and control both adjacent and remote APs. Figure 15 illustrates a possible multi-host physical organization for the logical structure represented by Figure 14.

A number of ideas from the MSG interprocess communication (IPC) mechanism [21] of the National Software Works have been generalized for use in organizing and addressing processes in the general purpose programming environment provided by ELAN. An important problem is obtaining the name of the process to which a message is addressed along with binding the process name to the

NOS Study

ELAN PROCESS HIERARCHY

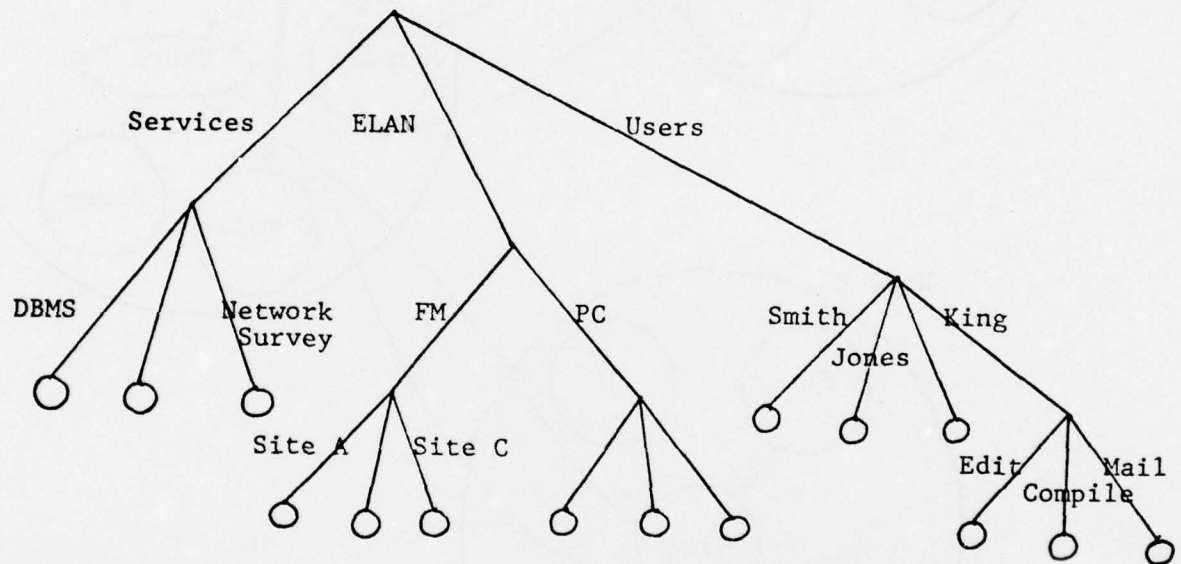


Figure 14. Site B

Logical representation for a typical ELAN process hierarchy. Three major classes of processes are illustrated: network services, ELAN kernel processes, and user application processes.

NOS Study

DISTRIBUTION OF PROCESSES IN ELAN PROCESS HIERARCHY

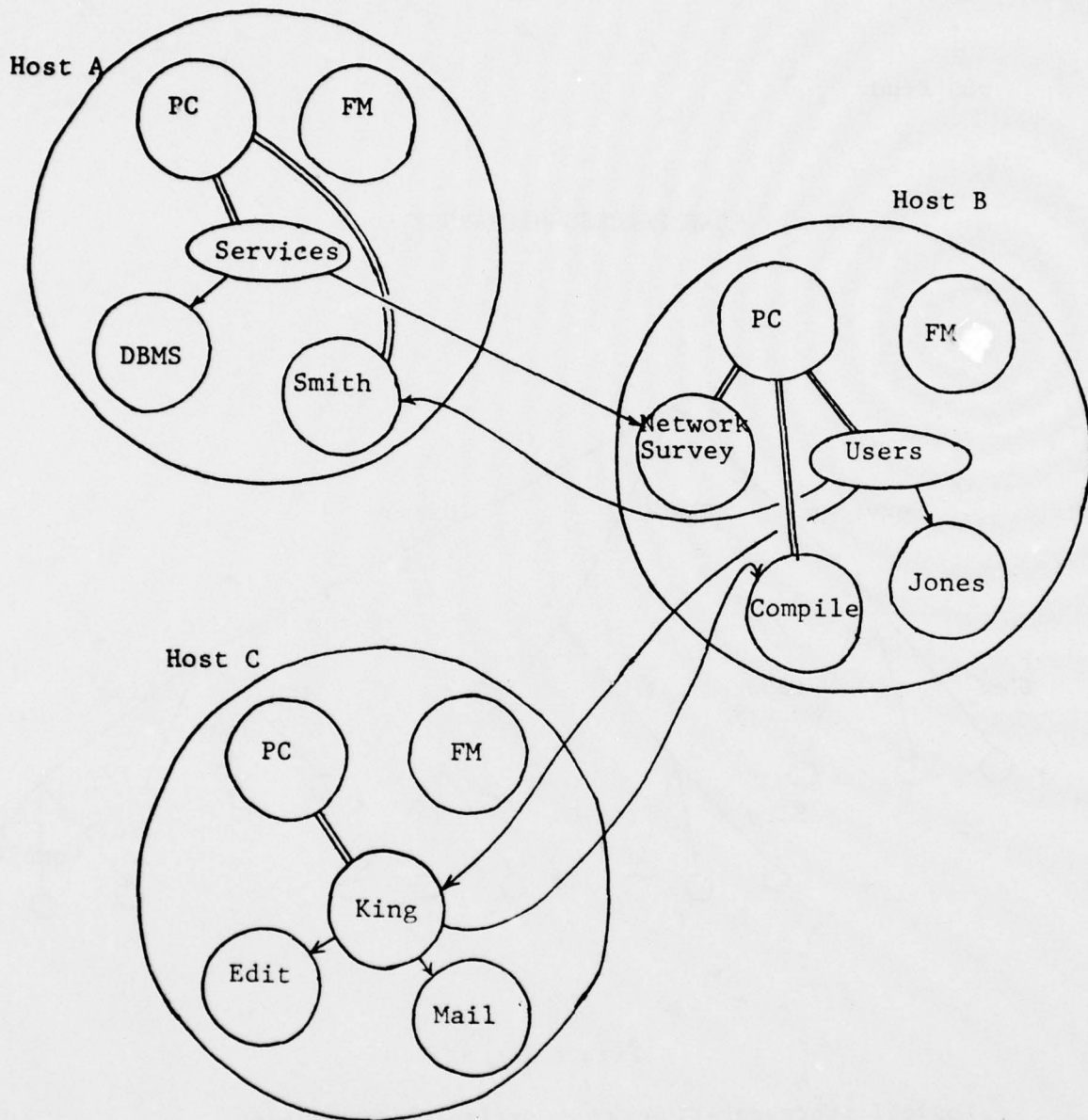


Figure 15.

One possible physical distribution of some of the processes from Figure 14. The single line connectors indicate the logical hierarchical relationships. The double line connectors denote physical control paths as exerted by the distributed process control agents.

physical location where the process exists. This problem is emphasized in a distributed system such as ELAN. Some of the notions presented in this section have been influenced by the work of Reed [78].

There are several equally important attributes that determine the name by which one process may view another process: physical location, ownership, function, and at times by some combination of these three attributes. Examples of these attributes include; "all processes at host A", "Smith's process", "the ELAN Process Control process", "the mail receiving process for Jones" and "the file transfer process at host A." In addition, the same process may have several attributes and therefore several different names.

The use of distributed components introduces the possibility that a given process will not be directly accessible to all elements of the interprocess communication facility. Instead, messages to the process may have to be routed to the site where the process actually resides and presented to the process by the local IPC agent. In a sophisticated system, a process could move from one site to another, retaining its logical name but changing its physical site address and local site process identifier. Thus, it is necessary to establish and maintain a flexible

binding between the logical name by which a process is known and its location dependent names and attributes.

Unlike MSG, process control (create, delete, stop and start) and interprocess communication (send and receive) are decoupled in ELAN. As a result, ELAN does not support the notion of a generically addressed message. Rather, a hierarchical functional naming scheme (to be described) is used to address messages to existing processes. When used in conjunction with a mechanism that enables a parent process to intercept messages directed to its children, functionality equivalent to MSG can be provided. A message addressed to a non-existent process is passed to its (future) parent which can create a process to receive the message. The original message will then be delivered to the newly created process. The advantage of decoupling the notion of interprocess communication from process control is that processes that wish only to exercise process control operations can do so, without having to induce such operation by means of IPC messages. In addition, the details of the coupling between process control and IPC can now be an application specific decision.

Processes in ELAN are organized in a hierarchy. Thus there are two separate resource hierarchies in ELAN, the file hierarchy and the process hierarchy. Like the ELAN file hierarchy, there

is a single conceptual ELAN process hierarchy to which every AP and KP belongs. Processes are addressed by a pathname through the process hierarchy.

As with the file hierarchy, we are faced with the problem of determining the physical location of a process from its pathname. One approach would be to use the same strategy used in a file hierarchy: distinguish between upper and lower portions of the hierarchy, replicate the upper part at each site and require subtree locality for the lower part. This solution is unsatisfactory for the ELAN process hierarchy. First, the restriction prohibits the creation and control of remote processes, an important capability. Second, the arguments about the frequency of updating that hold for the file hierarchy do not hold for the process hierarchy. Processes are the active elements in ELAN and can be expected to change state relatively frequently. Both of these points suggest that the penalty of interhost interactions is a cost that must be paid in order to achieve the desired flexibility.

To determine the physical location of a process from its pathname, the physical path implied by the logical pathname must be traced. There is a "speed up" mechanism that may be utilized to make this burden less of a performance penalty. Quite often

interprocess interactions (either process control functions or interprocess communication) occur as a series of events that occur within a relatively short time period. Also, a number of processes from the same host may interact with a given service process. Resolving the process pathname to a physical location for each interaction can be costly. Instead, the name resolution task can be carried out once and the ordered pair (pathname, host address) can be kept by the PC-KP in a local ELAN data base. When an AP issues a process control or IPC operation, this local data base can be searched for a match before expensive interhost interactions take place. The task of resolving pathnames which are only partially specified in the data base can be shortened by using the physical location of the matched "prefix" or parent process as a starting point for further name resolution.

The process name to physical location data base can be managed so that when there are no free entries, the least recently used entry can be erased to make room for a new entry. This data base is a distributed data base whose entries are in effect replicated at some of the constituent hosts of ELAN. Hosts that hold APs that reference the same remote process may all have entries that bind the name of the remote process to its location. At any time, the information in the data base may become invalid because of a change of state in one of the

processes for which ordered pairs exist. Rather than a potentially expensive duplicate data base updating scheme to maintain these data bases, a self correcting mechanism can be used. The nature of the name to location mapping operation is such that when a name/location association is discovered to be no longer valid, all that needs to be done to correct it is to clear the erroneous entry and reinitiate the operation.

The performance of this name look up mechanism will be determined by the number of times names can be resolved from the local data base rather than by interacting with all hosts along the path. This, in turn, is affected by the size of the table that holds (pathname, host address) pairs, the replacement strategy for these entries and the overlap of APs on the same host interacting with common processes. This problem is similar to that of memory management in a demand paging system.

There are several basic operations that can be performed on processes:

```
Create-Process(Process-Name, Program-Name, [Site-Location])
Destroy-Process(Process-Name)
Start-Process(Process-Name)
Stop-Process(Process-Name)
```

These operations allow a process to control inferior processes. Since the control functions apply to descendents,

AD-A056 078

BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS
NETWORK OPERATING SYSTEMS.(U)

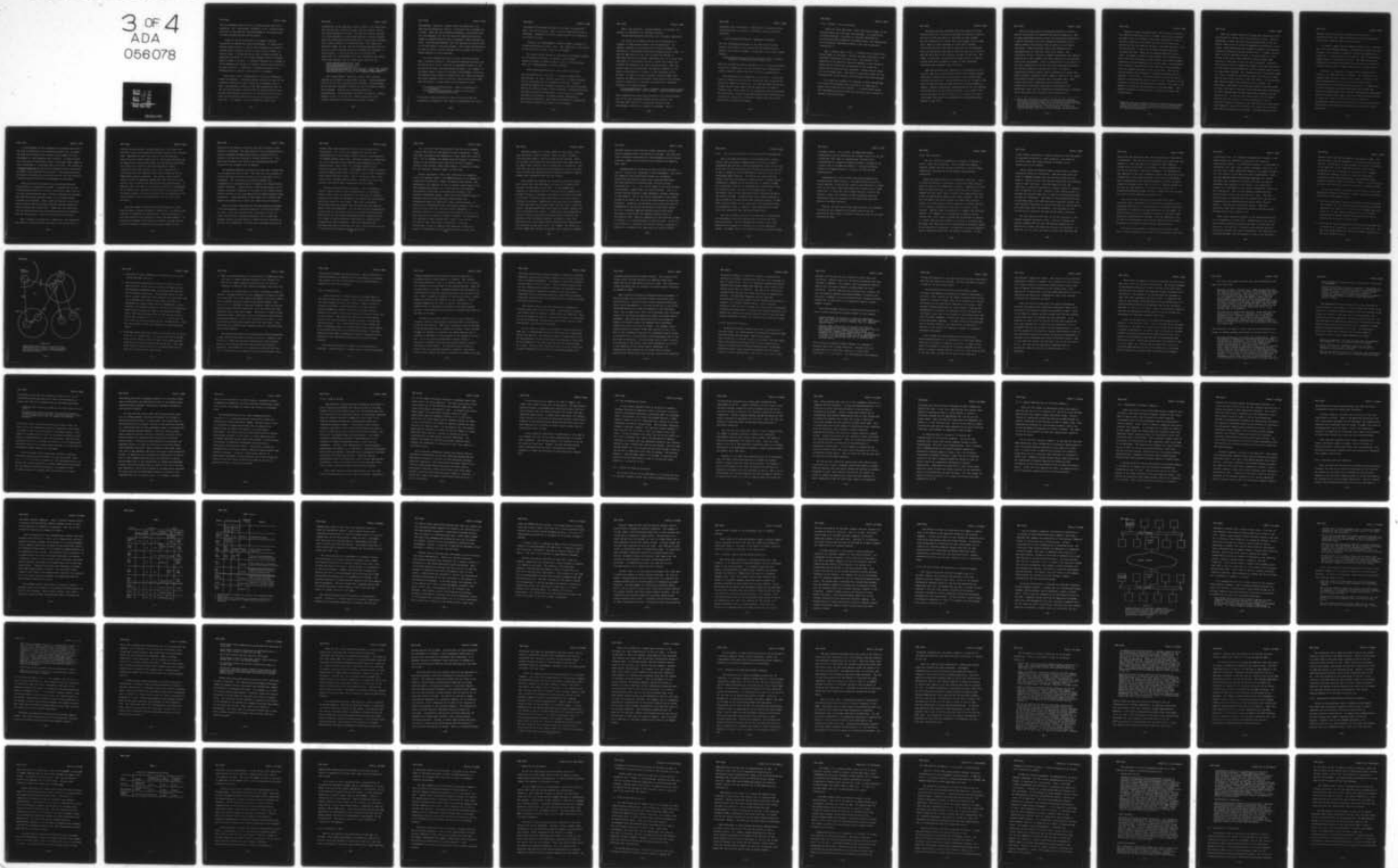
F/G 9/2

MAY 78 R H THOMAS, R E SCHANTZ, H C FORSDICK
RADC-TR-78-117

F30602-76-C-0425
NL

UNCLASSIFIED

3 OF 4
ADA
056078



only the pathname relative to the issuing process needs to be specified. The Program-Name parameter in the Create-Process primitive is the ELAN file system pathname of the program that the newly created process should execute.

A goal of the IPC primitives is to support flexible strategies for sending and receiving messages. An application program should be able to control which messages are allowed to consume buffer resources as well as select which messages it wants to read. A process should be able to specify groups of processes from which it will accept messages as well as message types it wants to receive. It should be possible for a sender to create messages that belong to different logical sets directed to the same receiver. In addition, it should be possible for a sending process to define a return port for messages.

These goals can be accomplished by a two part addressing scheme for messages. A message address includes the name of the recipient process and a port indicating the message class or type. Thus messages directed to the "New Transaction" port of the "Data Base Manager" process could be distinguished by that process from messages directed to the "Transaction 12324" port. Ports are structured objects that contain a unique part and a type part. For example, in the "Transaction 12324" port,

"Transaction" is the type part, while "12324" is the unique part. This two part port structure allows a process flexible control over which messages to receive. The type part allows a process to receive messages of a given type from any of several sources while the unique part allows selection of a given type of message from a given instance of a sending process. The unique part should be unique over all time so that delayed messages that are part of a broken dialogue do not get confused with messages of new dialogues. In the following list of primitive IPC operations, the Process-Name and Port parameters may contain values that signify a set of values rather than just one.

- Accept-Message(Send-Process, Port)
- Reject-Message(Send-Process, Port)
- Wait-Message(Send-Process, Port)
- Interrupt-With-Message(Send-Process, Port, Lower-Port, Routine)
- Take-Message(Send-Process, Port, Message, Sender, Reply-Port)
- Send-Message(Receive-Process, Port, Message, Reply-Port)

The Accept-Message operation controls which incoming messages are allowed to occupy buffer storage. Incoming messages that do not fit this description are discarded. Accept-Message may be executed repeatedly to build up a set of possible senders. Reject-Message retracts the permission granted by Accept-Message. Accept-Message specifies that when an incoming message arrives, it is put into a buffer belonging to the process, waiting for the application program to issue a

Take-Message operation. Between these two operations, the application program must receive notification that a message has arrived. There are two alternate mechanisms to accomplish this: one, Wait-Message, for the case where the application program wishes to suspend execution until a message is delivered and the other, Interrupt-With-Message, when the application process wants to be interrupted to process messages. This latter mechanism may be used to receive "alarm" messages from a process from which a normal message is being processed.

Ports are assigned priorities by the receiving process. When a receiver issues an Interrupt-With-Message primitive, the desired priority of the interrupt is expressed relative to the priorities of other interrupt generating ports. For example, if a process wishes to receive interrupts from messages arriving at two separate ports "Restart Transmission" and "Abort Processing" and wishes messages from the latter to take priority over those from the former, it would issue the following two calls:

```
Interrupt-With-Message(Sender, "Restart Transmission",  
                        0, Restart)  
Interrupt-With-Message(Sender, "Abort Transmission",  
                        "Restart Transmission", Abort)
```

The partial ordering defined by these two calls allows the processing of messages to the "Restart Transmission" port to be

interrupted by messages directed to the "Abort Transmission" port, but not vice versa. Thus, a partial ordering of ports is developed. Messages arriving at unordered ports are ordered by the time of arrival.

Finally, the Send-Message primitive sends a message to a receiving process at a given Port. The Reply-Port may be used by the receiver to send a reply to the message.

To allow a parent process to intercept the messages directed to its children processes, an optional last argument, Inferior-Process, is added to each of the six IPC primitives. Thus, the Accept-Message primitive becomes:

Accept-Message(Send-Process, Port, Inferior-Process)

The interpretation of this call is: accept messages from Send-Process on Port that are directed to Inferior-Process. All six IPC primitives are extended in this way. A parent process can examine all messages directed to an inferior process before the inferior knows of their existence. The parent process may take any desired action based on the arrival of a message directed to an inferior, including reading, modifying or destroying the message. This permits the parent to encapsulate the inferior process with respect to IPC messages.

When a new primitive, Release-Message, is executed, the message is released to its intended recipient:

```
Release-Message(Receive-Process, Port, Message, Reply-Port)
```

The effect of the MSG primitives can be achieved by a sequence of ELAN process control and IPC primitives. For example, consider the situation where a generic message is addressed to a previously unallocated Foreman process in NSW. In NSW, the MSG process on the site to which the message is addressed receives the message, determines that it is generically addressed to a non-existent process, creates the process and forwards the message to the newly created process. When that process replies to the message, the specific address for the newly created process is returned to the originator of the message. In ELAN, an overseer process, "Services.NSW", that controls the NSW service processes, would execute the following primitive operations to intercept all generically addressed messages to a Foreman process:

```
Accept-Message("Any", "Start Foreman", "Services.NSW.Foreman")  
Wait-Message("Any", "Start Foreman", "Services.NSW.Foreman")
```

When a message from any process directed to the "Start Foreman" port of the "Services.NSW.Foreman" process arrives, the "Services.NSW" process is interrupted and may use the Take-Message primitive to examine the message. Once it

determines that the message is addressed to a Foreman and that the Foreman must be created, it executes a process creation primitive:

```
Create-Process("Foreman-N", "NSW>Support>Foreman")
```

The file "NSW>Support>Foreman" in the ELAN file hierarchy contains the program to be executed by the Foreman process. Finally, the original message is redirected to the newly created Foreman:

```
Release-Message("Services.NSW.Foreman-N", "Start Foreman",  
Message, Originating-Process, Reply-Port)
```

When the "Foreman-N" process responds to the Originating-Process through the Reply-Port, the IPC mechanism will automatically return a specific address and port for future communication.

The advantage of this approach is flexibility. Notice that there is nothing in the primitive mechanisms that prevented one service to respond to generically addressed messages one way and another service, that shares the IPC mechanism to respond a different way. Also, there are no assumptions about the location of the services; a generically addressed message could cause the creation of a process on a different machine than the service process that fielded the message.

4.4.6 Process - File Interaction

There are three additional issues that have an impact on the characteristics of both files and processes: file sharing by multiple processes; providing an efficient mechanism for accessing the most frequently referenced files; and compensating for the physical separation between a file and the process referencing it.

When a process opens a file, it requests one of several well defined modes of reference: reading, writing, executing, or any meaningful combination of these three. The success of the completion of the Open-File operation depends on the outcome of access control checks and the existence of any concurrent conflicting accesses by other processes. Access control issues will be discussed later. The issue of concurrent access will be discussed here in the context of distributed files and processes.

In a single-site system, file sharing is often accomplished by allowing the processes direct access to the same area of memory which holds the file contents. In a distributed system like ELAN memory sharing schemes are impossible for processes on separate constituent hosts.

Providing multiple processes with simultaneous read and execute access to a file can be achieved by creating a copy of the file on each host where an accessing process resides. Multiple processes on a single host can all read from the same copy. When all such processes on a host have finished referencing the file, the copy on that host can be discarded. Since no alterations were made to the file there is no need to pass any data back to the original instance of the file. Any number of processes can successfully share a file in read or execute mode with no apparent change in their accessing efficiency over exclusive use of the file.

When two processes are organized to simultaneously write into the same file or one process writes while another process reads from a shared file, this multiple, uncoordinated copy solution does not work. The processes may not be located at the same site and thus may not have directly accessible memory in common. Without coordination, modifications to the file made by one process will not be seen by the other. As a result, for remote processes to share a file with at least one of them in write mode, there must be a mechanism for coordinating their updates to the file.

There have been several mechanisms proposed to provide a general distributed file updating and sharing capability. Most of these mechanisms provide a means for arbitrating references to a shared distributed file and exchanging information about updates between the multiple sites at which a file is being referenced. These mechanisms tend to be complicated and implementations for them are likely to be costly. The concurrency control required to support distributed file updating and sharing is more complex than that required in a centralized, single-site environment. This is primarily due to the effect of non-local communication and the cost of distributed algorithms for gaining consensus on global decisions. As long as communication delays between remote processors are longer than delays between a processor and its memory, there will be a speed differential between accessing local and remote data. When several hosts which communicate over relatively slow communication lines must participate to coordinate file operations, there will be significantly larger delays in reaching global decisions versus strictly local decisions (1).

-
1. While the efficiency of global file sharing and updating mechanisms is significantly less than single-site strategies, such mechanisms may be useful for reasons of reliability, load sharing and matching the requirements of inherently distributed applications. For the present, we feel that special purpose concurrency control schemes for distributed

Because of these considerations, restrictions are placed on the concurrent sharing of files in ELAN. Two or more processes may open a file for reading or executing regardless of their location relative to one another and the file as long as no process has the file open for writing. A process may open a file for writing only if there are no other processes currently accessing the file. These are the general ELAN file access rules. To support slightly richer process-file configurations which would be prohibited by these general rules, ELAN includes additional location dependent file access rules. One relaxation of the general restrictions on file sharing is that processes that are adjacent may share a file in any mode. This can be allowed because the techniques of single-site sharing can be employed in such cases. This more liberal sharing rule can be applied to processes that wish to share a file, regardless of its location, as long as the processes are adjacent. This rule is an example of visibility of distribution in the ELAN model. The physical location of resources can, if desired, be taken into consideration.

updating that are tailored to the exact needs of applications, rather than general purpose schemes, are the most efficient methods of providing some forms of distributed file sharing and updating functionality.

There are several patterns of usage that characterize most applications of non-exclusive write access to files. Several mechanisms in ELAN, based on these patterns, make up for the general restriction on location independent non-exclusive write access. These mechanisms are intended to support the functions typically provided by writably shared files. A common reason to open a file for sharable writing is to gain access to a lock stored in the file so that exclusive access to some other resource or some part of the file can be achieved. Exclusive write access, which is supported by ELAN, allows the arbitration on file access to be performed by the file system rather than by the application program. The information about the current usage of a file is stored in the directory entry for the file, rather than as part of the contents of the file itself. This accessing mode does not handle all instances of file locking. In particular, cases where for efficiency reasons a process wishes to keep a file open in sharable write mode and only locked during the short periods when exclusive access are required, are not handled by it. To provide such functionality and to support applications that require a locking mechanism to control concurrent access to resources other than files, the ELAN system could provide a class of lock objects for which fault tolerant distributed access could be provided. Multiple copies of lock

objects could be viewed as replicated data items and a distributed data base updating algorithm, such as developed by Thomas [39], could be used to maintain them.

In another common usage of shared writable files, processes use a file to exchange messages. The principle reason such usage develops is the absence of adequate interprocess communication mechanisms. On many hosts the only general form of interprocess communication is through shared files. The ELAN interprocess communication mechanism described earlier obviates the need for writably shared files to support most types of communication.

Finally, it is possible to support data synchronization services for ELAN application programs. The intention is to separate these services from the file system, so that the file system implementation does not incur any inefficiencies associated with distributed data synchronization. Schemes such as those developed by Johnson and Thomas [40], Thomas [39], Mullery [42] and Alsberg [43] could be implemented as application processes running under ELAN to provide synchronization services to other APS. These services could use the ELAN file manipulation and process control features to create higher level distributed synchronization and data base concurrency control mechanisms.

A second aspect of the interaction of processes and files is the need to establish a file referencing context for a process. The ELAN file hierarchy will contain a large number of files catalogued in a wide spanning tree structure. A single process will typically be interested in only a relatively small number of files catalogued in a localized area of the tree. The concept of a working directory has evolved for single-site file hierarchies to support such localized file activity. The working directory designates a directory in the interior of the tree that serves as a starting point from which file pathnames are expressed.

There are several new aspects to the working directory concept that are introduced in an NOS. A process may use the Create-Process primitive to create a new process remote from it. This new process needs to have a context from which to reference the ELAN file system and most often it is the same working directory as its parent. Thus, either the parent or the child must be remote from the working directory, and the problem of remote data sharing arises again. Assume for this discussion that the parent process and its working directory are adjacent; thus the new process is remote from its working directory.

When a process is created at a remote site, the contents of the parent's working directory are sent to the remote site to

provide the new process a working directory. Previously the difficult case occurred when two processes shared a file in write mode. Applying the same logic here, we see that process operations that require read access to the working directory can be supported by the directory copy, but that process operations that modify the directory (such as Create-File) require interactions with the parent's host. This limited form of distributed data base updating can be easily supported by the ELAN process control mechanism. The process control kernel process (PC-KP) of ELAN must communicate with the PC-KP at the parent's site to achieve the working directory modification. In addition, while the copy of the working directory is outstanding at the remote site, any modifications made to the directory by the parent process or any other process must be forwarded to the child process's site for incorporation into its copy of the directory.

The final aspect of process and file interaction is the means by which a process accesses a remote file. In general, for a process to reference the contents of a file efficiently, it must have access to the contents in local high speed memory. In single-site systems, files stored on secondary storage such as disks are typically transported into primary memory for the

duration of the access to the file and later returned to disk storage, if modified. The same pattern of file transportation to locally accessible primary memory must be carried out for a process in an NOS that accesses a remotely stored file. File data must be moved to the site of the accessing process and returned to the original site if updated.

One possible approach to moving file data from a remote host is to transmit the entire file. For files that are accessed as streams and where the entire stream is to be accessed, this is reasonable. For files that are accessed randomly and incompletely, it may make sense to transport only those portions actually accessed. A major factor in determining whether such a "paging" strategy is feasible is the speed of the communications network relative to the rate application processes generate "page exceptions". Recent advances in high speed local networks [60] suggest that paging over a local area network may be feasible.

Since the constituent hosts of the ELAN system are assumed to be heterogeneous, a problem arises due to differing formats for data. As we have presented the discussion of access to remote files so far, a process accessing a file of data has complete control over the manner in which the bits contained in the file are interpreted. To enable any reasonable handling of

multiple data representations, some of this freedom must be relinquished. One of the first examples of this was the standardization of the "Network ASCII" character representation for ARPANET TELNET connections. Processes wishing to communicate over such connections must translate their messages into this representation. Thus, there is a standard format for the "character stream" data type. The ARPANET File Transfer Protocol builds on TELNET connections to provide a service that transports files between different sites. Thus a second data type, "text files", has a standard Network format. Beyond that, little has actually been agreed upon, implemented and used widely.

We have not devoted much attention in the development of ELAN to issues related to data transportation or translation because these issues are potentially so large that it requires exploring the system in considerably more depth than was feasible within the context of this study. The issues surrounding distributed data soon develop into variations on the issues of general data base management. For the rest of this discussion we present some of the issues related to accessing structured data bases stored on distributed heterogeneous host computers but do not attempt to describe specific mechanisms for handling data transportation and translation. When such mechanisms are developed and added to ELAN they will most likely be built on top of the primitive mechanisms already specified.

The experience with "Network ASCII" and FTP in the ARPANET suggests that there are two aspects to data translation: atomic data type translation and composition of data items into files of data. For the ARPANET, the TELNET protocol defines a translation scheme for the character data type while the File Transfer Protocol (FTP) defines, among other things, a translation scheme for the structure (however simple) of text files.

Several approaches to data type translation are apparent. First, the approach taken in the TELNET protocol is an example of translation to and from a common intermediate data format. The advantage of this approach is that to become a participant in the data translation protocol, all a site must do is to develop translations to and from its internal data formats to the standard intermediate format. Quite often for different types of machines that are already partially standardized, the intermediate format is identical to their internal format. A problem with intermediate format translation schemes is that they tend to be less efficient than more direct approaches. A different method is to translate from the specific format of one host directly to the specific format of another. An advantage of this approach is that similarities in data formats can be utilized for more direct data translations. Its major disadvantage is that it requires that different translation schemes be implemented for every different pair of host types.

Whichever approach is taken, additional data types, other than characters, must be handled. These include: integers, floating point numbers, boolean values and bit strings. There are problems associated with these additional data types that do not appear when character string translation is considered. The difficult ones are associated with incompatibilities in the word sizes of different machines and the resultant problem of mapping complex data representations into one another.

Once methods for defining and translating different atomic data formats have been developed, the issue of translating the structure of data must be addressed. In the ARPANET FTP, sequences of characters form text files which are assumed to be multiple strings of characters followed by a carriage return and a line feed. Different constituent hosts store text files as different structures: TENEX hosts store them just as a sequence of characters with the carriage return and line feed in the file itself; IBM System/360 hosts store them as multiple records, one line of characters per record. In addition to the characters in the file, there are pointers and words counts to maintain the sequence of the individual character strings as a text file. Text files have an extremely simple structure when compared with the structure of many data files. For example, the format of a file on TENEX that contains relocatable code output by a compiler

contains blocks of data that may contain character strings, machine language code or relocation bit strings. The files that store information maintained by a data management system contain even more complex data structures with pointers and composite data items.

Standardizing the structure of data files has long been a topic of interest for researchers in data management. Two issues are emphasized when distributed systems are considered: translating structures of atomic data items and developing efficient methods for accessing small portions of large structured data files. The central aspect of the first problem is developing a common model for structured data. This is a current research problem in data management systems, regardless of distribution. One of the natural approaches to the second problem is to change the view of the relationship of files and processes. Currently, the prevalent view is that data must be transported to the site of the process for efficient access. This is a reasonable approach because current tasks being performed in distributed environments typically work on unstructured data. When the complexity of the structure of data is greater, it is less attractive to move the data. It may be more effective to introduce the notion of moving a process or a subtask of a process to the sites where the data is stored.

4.4.7 Use of the Primitives for File and Process Manipulation

While the ELAN primitives for file manipulation, process control and interprocess communication are useful for newly programmed applications, existing applications, with calls to the primitives of the constituent host on which they run, as yet have no way to expand the set of addressable resources to the set of distributed ELAN resources. To provide such addressability without requiring reprogramming, ELAN provides a general purpose encapsulator, modeled on the encapsulators in RSEXEC and NSW. One encapsulator needs to be programmed for each different type of constituent host. The job of the encapsulator is to reinterpret calls to the constituent host's operating system in the context of ELAN. It must provide functionality similar to that provided by the constituent host operating system. To make ELAN resources appear as the resources of the constituent host, the encapsulator must, in general, issue combinations of ELAN calls and constituent host operating system calls.

The ease of construction of an encapsulator is affected by the programming interface supported by the constituent host operating system. For the encapsulator to work at all, some method must be present to intercept calls to the operating system. On TENEX, calls to the operating system are made through

a transfer vector. As a result, the TENEX NSW Foreman encapsulates tools by redefining the transfer vector for the tool in question, with calls to encapsulation routines being reinterpreted by a separate trapping process to accomplish the intended task in the NSW environment. A similar mechanism, without the trapping process, is used in the Multics NSW implementation.

Encapsulation is not the only way existing application programs access ELAN resources. For applications that can be easily modified, it may be more effective and efficient to change calls to the local's operating system into direct calls to ELAN primitives. In this way, use of ELAN resources is direct and explicit, rather than indirect through a possibly heuristic translation from the interface of the constituent operating system to the ELAN interface.

Finally, the best use of ELAN resources from the standpoint of using the full capabilities provided would be by new applications whose model of resources from the start is the model presented by ELAN.

4.4.8 User Interface

The user interface encompasses all aspects of ELAN with which a human user encounters. This includes the command language, typing and line editing conventions, resource naming conventions, and resource access control and accounting conventions.

There are several goals of the ELAN user interface. First, it should be possible to establish contact with ELAN from any point of entry to the communication network. Regardless of their means of network access, all users should be able to issue a command that will place them in contact with ELAN without first having to authenticate themselves to another system. Thus they can view ELAN as the single system with which they interact. Second, it is desirable for ELAN to present a uniform interface to all resources regardless of the type or location of the resource. Special cases are difficult to remember and should be avoided. For example, if it is possible to specify access controls for a file, then it should also be possible to specify the access controls for a process or a device in a similar way. The model that users have of the organization of resources should be the same for all resources. In addition, as much as possible, similar resources should have the similar attributes. A user

should express references to remote resources in the same manner he expresses references to local resources. The naming of resources should not change because of changes in the connectivity of the network.

Third, provision should be made to allow users to choose alternative styles of interaction. Some users may favor one style of command language and line editing conventions over another. Different users have previous experience with different types of user interfaces. For many aspects of the user interface, it should be possible to support a style of interface that is familiar to a user. Finally, a user should be able to proceed with desired tasks even if some of the constituent hosts of ELAN are unreachable. The nature of the operations that may be performed under such conditions may be severely limited, but there should be some form of service available as long as some of the constituent hosts are reachable. The interface to this reduced service should be the same as the normal interface.

The most explicit embodiment of the user interface is the command language interpreter (CLI). This is a program that accepts commands from the user and executes ELAN primitive operations to change the state and structure of resources. In addition, the CLI causes programs to be put into execution and

thus allows the operations that can be performed on resources to be extensible. The CLI can run in a process on any host that can support it. With the advent of inexpensive small computer systems which can be part of a terminal, it may be possible to run the CLI directly in the user's terminal. The advantage of this approach is that very responsive service can be given to the program that is the user's entrance to the rest of the system. Other configurations that follow techniques currently in use on single-site systems, where the CLI runs in a regular process on the host are also possible. In either case, there is no predetermined binding between the site at which the CLI runs and the sites at which subsequent processes run.

One of the tasks that must be performed when a user initiates contact with ELAN is to establish the identity to be associated with the CLI. This identity, or principal identifier, represents the authority for whom operations are performed. An authentication service is needed to associate a principal with a previously unidentified CLI process. Once the identity of the CLI has been established, the principal identifiers of created processes are derived from this initial authentication.

One site in the network is identified as the user's home site. Several possible reasons exist for associating a user with

a particular site. For corporate organizational reasons, a user might normally be limited to a particular site that the organization controls. For efficiency reasons it is desirable to concentrate the files a user normally accesses at one site. In any case, when a user identifies himself to the system through a CLI, the location of his home site must be determined. Since a user may approach ELAN from different geographic or logical locations, the information concerning the home site must be accessible from anywhere in the network. For reliability reasons, it is desirable to have several alternative sources for this information. Thus, we can imagine an additional NOS function which returns the attributes of a user. This NOS function might be combined with the authentication service and implemented by an Authentication/Attribute Kernel Process (AA-KP). Possible user attributes include: home site, alternative home sites, authorized accounts, profiles used by programs such as CLIs, editors, etc.

When a user identifies himself to the AA-KP through a CLI, he is connected to his home site. Assume that the CLI runs as two processes distributed between the user's terminal and the home site. The terminal CLI handles such tasks as character echoing and command line typing conventions. The home site CLI exchanges messages from the terminal CLI and is the interface

between the CLI and the programming environment of ELAN. The user interacts with all subsequently created processes through the CLI. The division of tasks between the CLI and the process at the home site is not fixed and could vary from little processing in the CLI and most in the home site to command scanning and argument acceptance in the CLI and program initiation by the home site process. A process may create inferior processes either at the site on which it is executing or on other sites that have resources (processes) managed by ELAN.

Let us now consider an example of establishing contact with ELAN and the control of processes in ELAN. Referring to Figure 16 and the circled numbers there:

1. The user establishes contact with an AA-KP at site B through the terminal CLI. By suitable password exchanges, the AA-KP determines the identity of the user and the proper home site (site A). It then sends a message to the local Process Control Kernel Process (PC-KP) requesting that an AP be created for the user at his home site.
2. The PC-KP at site B interacts with the PC-KP at the home site to create an AP (labeled A) is created at the home site. The home site PC-KP returns the name of the newly created AP to the local PC-KP at site B.

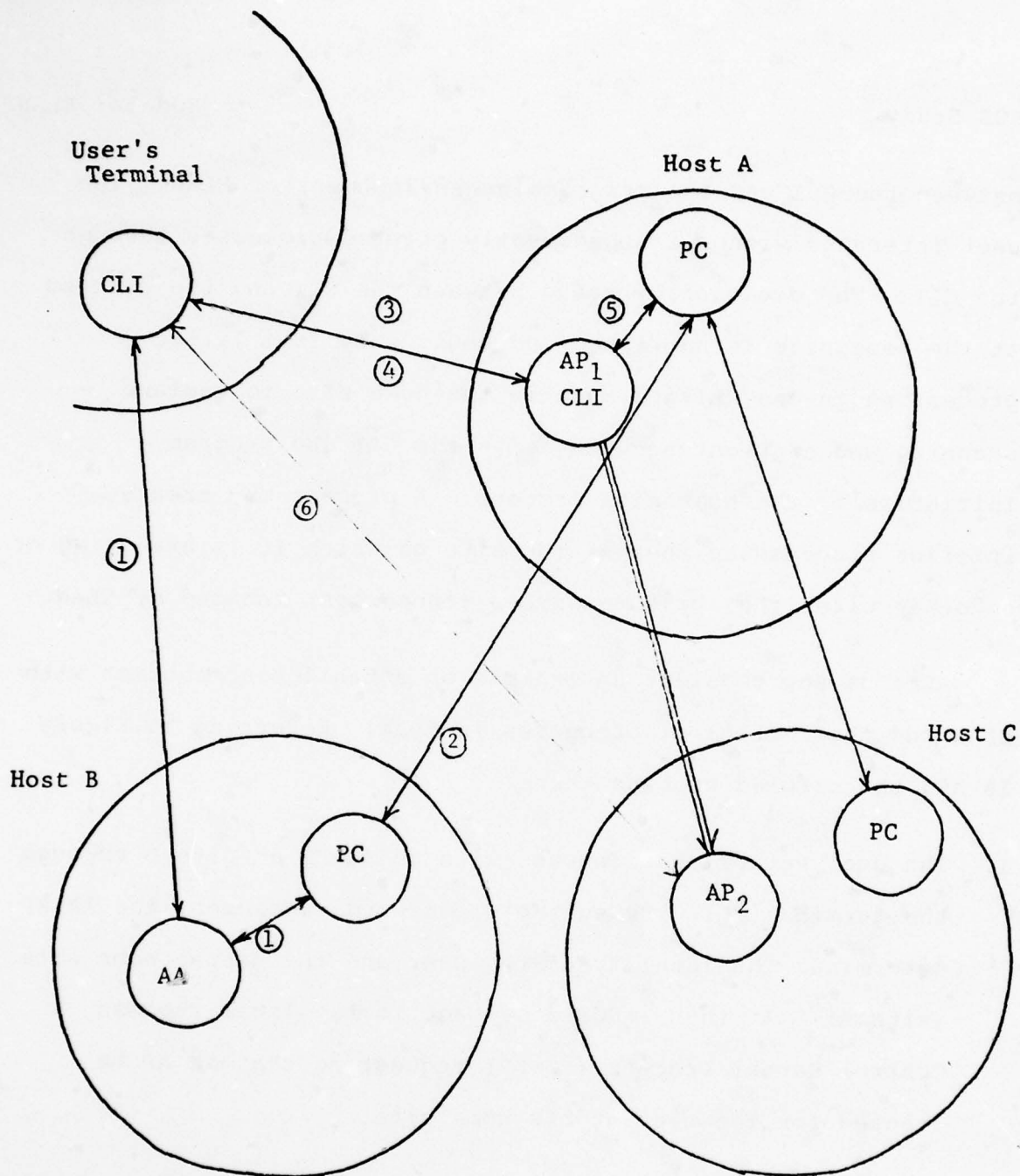


Figure 16.

Interactions that occur as a user logs into ELAN and begins to create a process hierarchy. The circled numbers refer to the discussion on pages 209 and 211.

3. The PC-KP at site B reconnects the terminal CLI to an AP running the home site CLI.
4. When the AA-KP requested the creation of an AP, it also specified that the AP initially execute the home site CLI. The messages exchanged between the terminal and home site CLIs are simple and concise. As mentioned above, we have split a normal single-site user interface into two parts: the CLI is designed to present a good human interface; it implements all typing conventions and command and argument scanning. The program running in the top level AP presents a good process interface to ELAN; it can execute all ELAN calls. The division between the CLI and the top level AP that we have presented could be shifted so that either the CLI or the top level AP assumed greater responsibility. From the user's point of view, the appearance of ELAN would not change.
5. An AP may create other APs, either at the same site or, as in this case, at another site. To create a process at another site, the AP interacts with the local PC-KP requesting the creation of an inferior AP at a particular site. The local PC-KP interacts with the PC-KP at the appropriate site to cause the creation of the inferior AP.

6. When a program executing in an AP desires to communicate with the user, it sends a message directly to the terminal CLI. The CLI can label messages from APs as well as restrict attention to specific APs, buffering messages from other APs.

Finally, an integral part of a user's interface to ELAN is a profile. Although profiles are not fundamental to ELAN as an NOS in the way they are to ATF systems and RSEXEC, they are so useful that any modern operating system design should support them. Since ELAN can potentially span many different hosts and provide access to many different subsystems, there are potentially many parameters that a user must supply. The profile is used to store information about a particular user's attributes, usage patterns and desired defaults. Notice that the profile of a user is used by multiple processes that maybe remote from each other. Thus there must be a mechanism for creating copies of the profile and distributing updates.

The problem with many current single-site profile mechanisms is that they are difficult to extend to new uses. Adding a new default is a difficult operation and thus only those defaults that are initially perceived as being useful are typically supported. New defaults or new application programs that have sets of defaults must be added to new profile mechanisms, rather

than one well managed and extensible one. While distributed systems introduce no substantial new requirements for profiles, the success of a wide-ranging system such as ELAN depends on profiles for easy use by humans.

4.4.9 Access Control

Controlled access to ELAN resources is accomplished by associating with each resource a list of principal identifiers for users allowed to access the resource. When a process is created a principal is associated with it so that when the process attempts to access a resource, the access control mechanism can check its principal against the access control list of the resource. The feasibility and utility of such access control list (ACL) mechanisms have been proven in the Multics operating system. For a distributed system such as ELAN, the major outstanding issue is to insure that access control appears to be uniform across all of the constituent hosts. Principal identifiers must apply throughout the system, regardless of the adjacency of the referencing process and the resource being accessed.

Processes are assigned principals by an authentication procedure. Authentication is accomplished by the AA-KP examining

evidence presented by the creating entity and deriving a principal based on that evidence of identity. The creating entity may either be a user or a process. When it is a process, the evidence is the principal of the creating process. When the creating entity is a user, authentication of the new process is accomplished by examining information known to be private to the user. This is known as "logging in" and in most cases, a secret password is used as this piece of private evidence. If the password supplied by the user is correct, the process is authenticated by assigning a principal to the newly created process. The principal associated with a process quite often is the name of the user.

An entry in an access control list is just a string of characters which lists principal identifiers that are permitted access to the resource. Additional information such as the mode of access permitted may be included in an entry. When a process attempts to reference a resource, the access control list is examined to determine if the principal of the process is on the ACL. If it is, access is permitted to occur; if not, a protection violation occurs and the process receives an error indication to that effect. For resources such as files or devices that are accessed frequently, expensive, high level access control checks that must be performed in software are only

done when the process initially attempts to access the resource. Subsequent access control checks based on this initial check are done by hardware mechanisms. An example is the point at which a process opens a file for access. Once the access control list has been searched and verified to contain the principal of the accessing process, subsequent accesses to the file, such as individual read or write operations are controlled by hardware mechanisms that have been initialized with the information gained from the software check.

The access control lists for resources are dynamically alterable over the life of the system. Users and APs acting on their behalf have the ability to modify ACLs for resources. The ability to modify the ACL for a particular resource is controlled by the ACL mechanism. Without such discretionary controls, new applications are difficult to develop.

So far, there is nothing in the model of access control for ELAN that is substantially different from that for a single-site system such Multics. The principal impact of distribution is on the implementation of the access control mechanisms in areas such as the management of the data bases used by the access control mechanisms. The principal identifiers used in access checks must be usable regardless of the location relationship between the

accessing process and the target resource. This requires that access control be done explicitly by ELAN KPs rather than implicitly by the underlying constituent hosts. The creation of unique principals and the integrity of access control lists must be managed by ELAN.

Most single-site operating systems provide environments in which processes can run protected without interference from outside sources. Thus the access control mechanisms described above can be implemented in a reliably secure way by the PC-KPs and FM-KPs which run in such environments at the constituent hosts. For a single site, there can be assurance that no process external to the kernel processes can subvert the access control mechanism. When multiple distributed sites are considered, however, the paths by which the separate kernel processes communicate offer a new area of concern. For example, when a remote process is created, PC-KPs must interact to set it up. Part of the information that must be transferred from the PC-KP of the parent site to the PC-KP of the new processes site is the principal of the parent. This principal will be used to derive the principal of the child. Since this information is critical to the operation of the access control mechanisms, the communication path between the two PC-KPs must be secure. Communication networks have only recently received the attention

of security studies. A difficult problem in communication security is reliably verifying that the source of a message is who it claims to be. This is an important problem because the communication network supporting ELAN will carry traffic from sources other than the ELAN KPs. The idea of an "electronic signature" has evolved from recent work [54] in communication security. An electronic signature has the important characteristic that its validity can be reliably determined by a computer in a straight forward manner. Such electronic signatures could be used to support the access control mechanisms of ELAN by providing means to ensure that messages directed to an ELAN KP were indeed from another ELAN KP and not from some other process attempting to mimic a KP and break the security of ELAN.

4.4.10 Reliability Measures

Developing reliable systems that recover from errors is a difficult task. Our investigation of reliability issues for ELAN centered around two questions: What is the current state-of-the-art in reliable single-site systems, and what impact does a distributed architecture have on the reliability of a system? Previous work has concentrated primarily on hardware reliability and on error recovery techniques that are finely tuned to specific applications. The thrust of the work in the

hardware area has been on error detection rather than error correction. Recently some error correction mechanisms have been included in hardware. For example, some semiconductor memories employ error correcting codes to provide single bit error correction. Specific applications for which error recovery has been successfully demonstrated beyond hardware reliability are electronic telephone circuit switching systems, message switching systems and, in some cases, file systems for general purpose computer systems.

Well defined applications are typically more successful at error recovery than general purpose systems for two reasons:

- Access patterns are limited in a specific application. Thus, the number of possible failures and, more importantly, the effect of those errors is smaller than in a general purpose system.
- Specific applications that are reliable can often be characterized as transaction oriented in the sense that the main computation is a composition of many small transactions. Examples of transactions include single telephone calls and single packets of data. Recovering from a failure to an operational state can be accomplished by discarding all transactions that were in progress and starting afresh.

Error recovery in general purpose computer file systems is typically done with a single objective: returning the file storage space to a state that conforms to a legal state understood by the file system. The shortcoming of this approach

is that the semantics of any application program data bases built upon the file system are ignored; the only consistency insured is that of the file system itself.

The distributed architecture on which ELAN is based influences the approach to designing reliability mechanisms in two ways. First, because of the number of potentially failing components involved in ELAN, there is an obligation to take the reliability of the distributed components into account. When multiple failure-prone parts are used together, assuming the independence of failures, the probability that all parts are functional at any time is the product of the probabilities that each individual component is functional which is less than that of any individual component. A usable system can be developed only if failure recovery, and particularly the ability to function in the presence of component outages, is integrated throughout the system.

For a system that has processes and files which are partitioned physically it is reasonable to think about taking some real-time action in response to errors. It makes little sense, for example, to develop an elaborate mechanism for one process to repair another process if they will both normally fail at the same time, as would be the case in the crash of a

multiprocess, single-site system. The thrust of the reliability mechanisms in ELAN is to improve the reliability of NOS services that require resources distributed among multiple sites rather than to improve the reliability of the single sites. The expectation of being able to provide robust NOS service relies on separation and statistical independence rather than inherent reliability of individual processes.

There are several places where the natural autonomy of resources provided by the single-site constituent hosts can be preserved in the primitive operations provided in ELAN. First, in the ELAN file system, we have already seen that the autonomy of the constituent file systems has been exploited. It is possible to reach a file as long as the site that catalogs the file (and, possibly the second site that stores the file) is reachable from the site at which the referencing process resides. For the purposes of the ELAN file system, the failures of some of the constituent hosts do not effect the ability of processes to reference files catalogued and stored on different hosts. Thus, for the ELAN file system, the individual file is the autonomous entity. The failure of reaching one component does not necessarily imply the failure of reaching other similar components.

While files are passive resources, processes can take a much more active role in reliability mechanisms. The natural autonomy in the process structure of ELAN is located at the point where interactions occur between the possibly multiple processes of an application. The failure of one process does not necessarily imply that other processes have also failed. In addition, the state of one process can be protected from the malfunctions of a second process by means of the physical and logical isolation possible in a distributed system. Thus, the two atomic resources that will be considered from the point of view of reliability mechanisms are individual processes and files.

Errors in the operation of an application program can be detected in two different ways. First, ELAN can detect erroneous conditions in the state of a resource and second, an AP can make a similar determination. In either case, an AP will be notified by ELAN or determine itself that an ELAN resource which it is using is in an erroneous state and that it must perform some corrective action. The AP can receive such notification in one of several ways: by a status code returned from a call to a primitive ELAN operation or by an IPC message or an interrupt from its parent or an ELAN KP that has detected the problem.

For files, the types of errors that can be detected by the NOS or by the AP are:

- Failures to reach a storage host. ELAN can determine when the host at which a file is stored is unreachable and notify the referencing process. This will have an impact in cases where direct contact with the storage host is necessary. If, for example, a process has opened a remote file for write access, and a copy of the file has already been successfully transmitted back to the referencing host, then as long as the file is kept open, the status of the storage host has no impact on the referencing host. When the file is closed and a modified copy needs to be delivered to the storage host, then corrective action must occur if the storage host is unreachable.
- Failures due to conflicting accesses. Given the rules for sharing files in ELAN, it is possible for an OpenFile operation to be in conflict with the use of the file by another process. This is intended to be a transient situation and appears similar to an unreachable storage host. A process that encounters such a situation must be notified so that it can try the file opening operation again at a later time.

For a process, the types of errors that can be detected by its parent or another process that wishes to interact with it are:

- Failures encountered in interprocess communication. When an AP attempts to communicate with another process the other process may be unreachable. This could be due to failure of the destination process, failure in the communication medium, disconnection of the remote host from the communication medium or scheduling delays in the destination host. Some of these failures will be detectable by the remote PC-KP and the sending AP can be explicitly notified of the failure. Other failures can only be assumed to exist because of the failure of any process on the remote host to respond. In such cases, error recovery mechanisms must accommodate errors that can occur in such conclusions: it is very difficult to distinguish between a process that will

never respond and a process that is delayed for a long time in responding.

- Failures encountered in process execution. The parent of a process or the process itself can detect a second class of errors that occur in program execution, such as access control violations, use of resources in excess of allocation, execution of illegal instructions or in general, illegal process states including illegal application specific states.

The appropriate corrective action for errors detected in file and process operations is to either negate the effect of the failure or to restore reusable resources that are involved in the failure to a state so that they can be utilized again. For files, there are two places for corrective action to occur: at the host where the application process that encountered the error resides and at the file catalog site where information is kept about outstanding references to the file. The application process must take one or more of the following types of corrective action to cause proper operation to resume:

- Retry the operation. For file failures that are transient, the passage of time can sometimes solve the problem.
- Try to reference an alternative source for the file. Concurrent access to such alternative sources must be managed by the application process.
- Back up the state of the AP to a consistent spot and abandon the part of the computation that is experiencing the error.

The FM-KPs at both the file catalog site and the site of the referencing AP must perform one or more of the following to insure that the file in question may be referenced once again.

- Close the file, leaving it with the its state before it was opened.
- Determine that the file storage site has been delayed in responding and assume that after a suitable period of time, it will respond and be able to complete the desired operation.

In any case, after the corrective action has been taken, the state of the file system will have been repaired so that future references to the file can proceed in the normal fashion. This means, for example, that if the file was being accessed in write mode by a remote process at the time of failure, then after the corrective action has occurred, the state of the file system will show that the file is no longer being accessed. It could then be accessed by another process in write mode.

As for failures in file operations, failures in process operations can be corrected either by an AP or by ELAN. Minor errors can be corrected by another AP. The malfunctioning AP can be put into correct operation via normal communication mechanisms and can execute instructions normally. An example of such correction would be a message sent from a parent to a child

instructing the child to abandon whatever it is currently doing and to reinitialize its data bases so that it can handle future requests without residual, and possibly erroneous information from previous requests.

For more serious errors, where the failing process cannot run in a normal fashion, the only alternative is for ELAN to perform corrective action. In most cases, the process must be destroyed and the environment in which it was executing restored to a consistent state so that subsequent access to resources that it was using may proceed normally. There are several places where corrective activity can occur including the hosts of the parent process, any communicating processes and the failed process itself. Corrective actions applied to one process can sometimes result in a chain of corrective actions that affect other processes and files. When a process is destroyed, every file that it had opened at the point the failure occurred must be closed. In addition, every interprocess communication stream on which it was communicating with other process must be closed. Any processes that it had created must be stopped and destroyed. Finally, the parent process must be notified that its child has been destroyed. One of the processes of an application can break the chain reaction of these ELAN procedures by taking on the responsibility for correcting errors. For example, a process

that has been notified of a child's demise, instead of blindly attempting to communicate with the child and encountering a fatal error itself, can attempt to restore the child to a reasonable state.

The mechanisms above do not offer substantially new capabilities over those provided by conventional single-site systems such as Unix, Multics or TENEX. The point here is that it is possible to employ mechanisms that are present, but underutilized in single-site systems, in a distributed environment. The reason they have been underutilized in making applications reliable is that there is little opportunity for autonomous operation within single site systems. The distributed architecture of ELAN makes mechanisms that could not be effectively utilized before useful. The extent to which the responsibility for failure recovery should be shared between ELAN and APS is unclear. It may be a non-trivial task for an applications programmer to make effective use of the reliability mechanisms and the extent to which they would be used at the application level is unclear at present.

4.4.11 ELAN as an NOS

ELAN provides a general purpose programming environment in which resources distributed over many hosts may be accessed. Despite the fact that ELAN is built on a distributed base from the resources of its constituent hosts, the underlying conceptual model or abstract machine it supports is very close to the abstract machine presented by a conventional single site time-shared computer system. A process may access network resources regardless of the relationship between the host on which it runs and the host on which the resources reside. The hosts that can support ELAN are heterogeneous. The system programs of ELAN perform suitable transformations on the resources of these heterogeneous hosts so that they are compatible. Finally, since ELAN presents a general purpose programming environment, it is extensible. Application programs whose needs are unknown to the NOS can be implemented from ELAN primitives to provide new services. Resources used by applications can be controlled by a user or by an applications programmer; there is no need to work around fixed decisions about resource allocation made by the operating system.

While ELAN contains several improvements over the other three systems presented, it has some shortcomings. Because of

all of the support processes necessary to implement ELAN, ELAN hosts need to be fairly powerful. The hosts envisioned as ELAN constituents are of the DEC PDP-10 and Honeywell 6000 class rather than smaller minicomputer hosts. At present such smaller hosts appear to be the better economic choice for many applications and will probably continue to be in the future. Along with the trend toward smaller computer systems, we expect to see an increase in the number of separate systems that must interact on a regular basis. As discussed in the next section these smaller machines are likely to need the support of larger ELAN-like hosts, but for a variety of reasons can not be integrated into an ELAN system as normal ELAN hosts. For example, the number of interactions between ELAN KPs would probably be overwhelming if there were a very large number of constituent hosts.

One of the main differences between the abstract machine provided by conventional single site operating systems and the ELAN abstract machine is the set of rules governing allowable ELAN file references. The restrictions that apply depend on the location relationship between processes and files, and are potentially confusing. These restrictions result from ELAN's attempt to provide uniform and efficient access to resources. A system that removes the complex restrictions would do so at a cost in efficiency.

A third deficiency of ELAN is its lack of support for higher level views of data and its processing. The ELAN abstract machine provides a programmer with unstructured data files and with processes that can execute machine instructions and make operating system calls. A higher level view would, for example, permit access to collections of structured records, rather than a sequence of uninterpreted bits and provide primitive operations which operated on the fields of these records, rather than on unstructured files of data.

A final potential shortcoming of ELAN might be that some of its concepts have not had an actual implementation to validate their soundness. The concepts appear to be feasible as far as the study can take them. However, the more detailed design effort associated with an actual implementation would be necessary to judge the difficulty of achieving such a general NOS.

4.5 The PC-ELAN System Design

If the current downward trend in the price of computer hardware continues as expected, within a few years it will be possible to dedicate substantial processing power to individual users. Each person in an office environment may have a small computer system to use as a dedicated tool, similar to the way telephones are used in offices today. One principal difference between a "personal computer" (PC) and most current systems is that the user will not need to share his computer with other users. As with telephone sets, there will be little concern if a PC is not in constant use. While the dedication of computer resources to a single user has undeniable benefits, the isolation implied by such a system must also be anticipated. PCs will need outside support for file sharing and long term storage as well as the ability to communicate with other PC systems. The Personal Computer - ELAN (PC-ELAN) model is an attempt to generalize the attributes of the ELAN design so that it can provide system support to a collection of PCs.

4.5.1 Purpose and General Attributes

The primary goal of the PC-ELAN model is to provide the user of a personal computer system with uniform expanded accessibility

to distributed resources as in ELAN, while preserving the advantages of his PC. A major attribute of a PC is that it can provide uniformly responsive service because there need be no activities within it that are not directly related to the user's computation. Unlike a shared computer system the amount of time taken to perform an operation will not vary. Thus, the user can develop a consistent model of the delay associated with a particular operation.

The low cost and single-user nature of the PC suggests that the number of separate systems included in an integrated environment would be much greater than in ELAN. The number of hosts would approximate the number of users. Thus, we would expect to see 10 to 100 times more individual hosts in a PC-ELAN system than in an ELAN system (assuming a typical shared computer can support 10 to 100 users).

Because a PC is under the complete control of an individual user, the small hosts participating in PC-ELAN will exhibit greater autonomy than the larger service hosts. For example, a user may disconnect his PC from the network and continue his computation accessing only the resources of the PC. When a PC is disconnected from outside communication the information inside it is secure since there is no way an outside agent can access the

data. This attribute may turn out to be a pragmatic solution to computer security problems. In large multiplexed machines, it is difficult to prove that data accessible to one process is inaccessible to others. Since a single-user machine may be disconnected from outside communication and its file storage may be physically removed from it when not in use, a much more convincing argument for the security of data can be made. This is essentially the current method for performing secure computing with the difference that currently, highly classified computations generally run alone on a large, expensive computer system. Concern for privacy is not limited to issues associated with classified computations. There is increasing concern about the privacy of data in normal day to day activities of individuals and business. A PC is capable of supporting the notion of private files that never leave the machine nor can be accessed by outside agents. This is useful for storing work that is sensitive or partially completed.

The PCs are well suited for applications that require close interaction with a user. Examples include text editors, message handling systems, data base query systems and interactive program debugging systems. When used in this way PCs can remove the burden of supporting applications requiring fast response for small computations from the large hosts, where the overhead of

providing such services often outweighs the small computation required by them. In our view, applications that require large amounts of processing power or large amounts of directly accessible storage are not appropriate for a PC but are for larger hosts. Thus, PCs and the large hosts can complement one another in a cost effective way; the PC can be used for highly interactive tasks and the larger machines can be used to provide relatively large amounts of computing or data storage.

An important goal of PC-ELAN is to preserve the responsiveness of the personal computer. One of the main problems with current time shared computer systems is their unpredictable, and often slow, responsiveness. This is largely due to demands placed on the shared computer system by a dynamically changing number of users. For a PC to display consistently responsive behavior, tasks that are not directly related to the user's requests should be limited in order to prevent them from consuming PC processing or storage capabilities. Experience has shown that if care is not taken operating system overhead can slowly grow due to the cumulative effect of providing more and more services. For this reason, an important design principle is to closely scrutinize new tasks proposed for the PC.

4.5.2 Typical Applications for Personal Computers

There are two classes of applications that would benefit from the use of a PC or a group of PCs. The first class includes applications that support highly interactive tasks. Current notions of interactive versus non-interactive applications are greatly influenced by the machines we have to support them. When something as different as a PC becomes available, some of the current paradigms of computing are likely to change. The result is that we will probably see many more applications viewed as interactive tasks.

Use of the term "personal computer" in the name for this NOS model should not limit consideration of potential applications to ones that employ machines dedicated to the tasks of a single user. A second class of applications suitable for PC-ELAN is cooperative use of a collection of small machines to accomplish a single task. Examples include process control applications and the composition of separate application systems so that information acquired and maintained by one is usable by another system. Unlike the first application class, the emphasis here is not on interactive capabilities but on interconnection.

4.5.3 Advantages of Personal Computers

There are several factors that make personal computers more attractive than the shared computer systems we have today. Most of these result from the desire to create computing environments that are more hospitable than can be provided by current systems. First, there is need for computers that give a uniform, fast response. It is difficult for humans to adapt to an environment where the system sometimes responds quickly and sometimes slowly, especially when this variability occurs over relatively short time periods. With a personal computer that is not shared, the variability of task completion times can be made to be negligible. The speed of the hardware for a small machine can be as fast for larger multiplexed machines. With both uniform and fast response characteristics, the personal machine should provide a computing environment whose responsiveness satisfies the user's desire for quick reaction to simple tasks.

A second reason for the attractiveness of a personal machine is its ability to support new and innovative techniques for communicating with the user. This has been one of the weaker aspects of the human interface of current computer systems. Few systems go beyond providing communications that can be supported by a teletype. While improvements have occurred in terms of

devices with increased speed, the underlying character- or line-at-a-time orientation of the user interface persists and is firmly entrenched in most application programs. As a result, most user input/output interfaces are one dimensional, while users tend to think in terms of two or three dimensions. Attempts at presenting or accepting two dimensional information are typically frustrated either by the difficulty of building a two dimensional communications interface on top of a one dimensional system support or by the high processor demands inherent with flexible display devices and with handling interrupts associated with sophisticated input devices. The first problem can probably be solved through programming, while the second is an intrinsic problem that can best be solved by dedicated hardware.

Personal computers can help in two ways here. The problem with high processor demand results from multiplexing processors among many users: one user's high demand degrades other users' service. Even with a separate, multiplexed "display processor" the number of users that can be supported is limited. With a personal computer, there will be more processor power devoted to each separate user. Displays can be driven either by the main processor of the personal computer or by a second, dedicated display processor that is part of the personal computer system.

The dedication of more processing power will make previously unacceptable processor demands more reasonable.

Personal computers can also help in the area of the user input/output interface. Since a new operating system and a new set of application programs will be developed, the communications interface can be designed from the outset to support the ideas of two dimensional input and output. Once a two dimensional interface is supported at the system level, application programs can more easily extend this view into their problem domain.

Thus, personal computers offer several advantages over larger time-shared computer systems. While this study is directed primarily towards NOS issues, some investigation of the attributes of personal computers has been necessary to determine their support requirements.

4.5.4 Existing Single User Machines

While the need and motivation for personal machines has been discussed, we have yet to show that these machines are feasible. The purpose of this section is to demonstrate that personal computers with sufficient computing power to satisfy the needs of the intended users are an economic possibility. Several existing systems that provide good estimates of the size and power of

realizable personal computers. Table 1 contains characteristics of several existing personal computer systems as well as some larger machines for purposes of comparison. The rest of this section will discuss the examples of Table 1.

Horn and Winston [65] have hypothesized a machine that they estimate could be built (in 1975) out of commercially available modules for about \$33,000. Their machine is a microprogrammable 16 bit computer system with 64 bytes of high speed cache memory, 64 kilowords of main memory, 256 kilowords of swapping memory and 10 megabytes of file storage. The processor has 16 registers, an arithmetic-logic unit and a microcode interpreter with memory. In addition, there is a user console with a keyboard, television monitor display, 256 kilobits of display memory and a hard copy output device. Communications with other machines is through a high speed communications interface. The processor, memory elements, user console and communications interface are connected by a bus so that any active element can communicate with any other element connected to the bus.

The proposal by Horn and Winston is stated in terms of hardware; there is no mention of the abstract machine seen by the user of such a system. They do state, however, that "Such a system has enough storage and processing power to satisfy the

NOS Study

TABLE 1

| System | Processor | | | | Memory | |
|----------------|-----------|-----------|-----------|-------------------|--------------------|-------------------------|
| | Registers | Word Size | M Program | Similar Machine | Main Size | Secondary Size |
| | # | bits | | | B: Byte W: Word | |
| Winston-Horn | 16 | 16 | yes | | 64KW | disk 10MB |
| Xerox Alto | | 16 | yes | Data General Nova | 48-64KW | disk 2.5MB |
| IBM 5100 | | | yes | IBM/370 | 16-64KW | tape 204KB |
| DEC 310W | | 16 | no | PDP11/20 | | floppy disk 128KW |
| MIT LISP | | 32 | yes | | 64-128 KW | disk 50MB |
| BBN LISP | | 16 | yes | PDP11/40 | 128KW | disk 50MB |
| TENEX PDP10-KA | 16 | 36 | no | PDP/10 | 256KW | 100MW |
| VM/370 | 16 | 32 | yes | IBM/370 | 500KW | |
| UNIX 11/45 | | 16 | no | PDP11/45 | 48KW | |

NOS Study

TABLE 1 (cont.)

| System | Devices Supported | | Hardware Costs | Remarks |
|----------------|------------------------------|--------------------|------------------|--|
| | Display | Hardcopy | | |
| | b: bit | | | |
| Winston-Horn | yes 256Kb | yes | \$33,000 | hypothetical machine |
| Xerox Alto | yes, high resolution T.V. | no | | |
| IBM 5100 | yes, very small | optimal, low qual. | minimum \$8,500. | price reported in another vendor's ad. |
| DEC 310W | yes, VT52 | yes, high qual. | | Low end of generality/power. Aimed at word processing market. |
| MIT LISP | | | \$35,000* | "Real time" LISP; 2 x execution speed of TENEX running 1 user LISP.* |
| BBN LISP | | | \$100,000 | Machine complexity similar to MIT LISP; different cost due to profit and support costs. |
| TENEX PDP10-KA | | | \$450,000** | Supports 5-30 users, single user cost = \$15,000 - \$90,000 virtual address space 256KW |
| VM/370 | | | \$2,500,000** | Supports 20 - 40 users, single user cost = \$125,000 - \$80,000 |
| UNIX 11/45 | | | \$50,000** | |

* No profit margin or software cost in price. Price expected to drop to \$25,000 soon.

** These figures from "a UNIX-based local processor and network access machine," E.G. Manning, et. al. Computer Networks (1976) vol. 1, pg. 139-142.

computational needs of most users now requiring access to a classical timesharing system." Given current prices, most organizations would have to view this system as part of a pool of systems that can be shared among a group of users. Presumably each would have file storage kept on a demountable disk that could be taken away from the machine. Anticipated reductions in price should make it possible to dedicate one such system to each person that needs one.

The second system to be considered is a personal machine being developed at the Xerox Palo Alto Research Center (PARC). The original version of this machine, known as the "Alto," has already had several successors. These systems are intended for an office environment. Lampson [66] describes a single-user machine that has a 16 bit processor, 48 to 64 kilowords of 800 nano-second main memory and a 2.5 megabyte file storage. The processor executes an instruction set similar to the Data General Nova minicomputer. The user console has a keyboard and an high resolution television screen turned so that it has the same format as a sheet of 8-1/2" x 11" paper.

The operating system for this machine is intentionally minimal. Instead of a "closed" system that protects application programs from the awkward appearance of hardware and from the

mistakes of other application programs that share the system, the Alto operating system augments the hardware, but leaves most of the base hardware accessible to application programs that may want to access it in unconventional ways. After these systems have been fully debugged and produced as tools for an office environment, they will not be programmed by unsophisticated programmers. Rather, a fixed set of application programs will be available to perform very well defined tasks.

Another class of systems under development are the so-called "Real-Time LISP Machines." Workers at the MIT Artificial Intelligence Laboratory have developed such a system, similar in motivation to the system described by Horn and Winston. Many interesting application programs have been written in LISP, but because of unsuitable execution environments, most run very slowly. LISP machines are designed to correct this situation by providing LISP primitives at very low levels of the system, if not directly in hardware. In the MIT system, the processor is a 32 bit machine built from a microprogrammable processor (also built at MIT) with a microprogram that can interpret LISP programs. Main memory is 64 to 128 kilowords and secondary storage is provided by a disk that can hold 50 megabytes of data. It is estimated that a LISP program will run on this system about twice as fast as it would on a DEC KA10 processor stand alone

under the TENEX operating system. A prototype machine has been built and a small number (less than 10) of second generation LISP machines are currently being designed and built. The cost of a second generation machine is estimated to be between \$35,000 to \$50,000.

A similar effort is underway at BBN to build a LISP machine based on a microprogrammable DEC PDP-11/40. A home-built paging device has been attached to increase the address space of the PDP-11. The cost of this machine is approximately \$100,000.

IBM has successfully marketed the IBM 5100, a desk top computer system that can execute APL or BASIC programs or operate in a communications mode. The 5100 has a self-contained keyboard, video-display and magnetic tape cartridge drive, as well as an optional hard copy printer. The processor of the 5100 is a microprogrammable machine whose program interprets the IBM 370 instruction set. The user interface to the 5100 is similar to the appearance of the APL and BASIC interpreters running on a timesharing system; the difference is that there is no outer command level interface where, for example, files can be manipulated. All interactions with the system must be done from within the context of the APL or BASIC interpreter.

Several companies offer word processing systems, some of which could be regarded as personal computers. One example is the DEC 310W, a word processing system with a video terminal and a high quality typewriter output device. The processor is a PDP-11/20 and the file storage is a pair of floppy disks, one to store documents and one for system storage. A single floppy disk can hold 128 kilowords (16 bits per word). The 310W may be used either in stand alone mode or communication mode. In stand alone mode, document preparation operations including normal text editing, document storage and retrieval from floppy disk, and listing of stored document in memory technologies can be performed. In communications mode, the 310W can act as a teletype device or as a remote job entry station.

Finally, there is an anticipated development that could have a significant impact on future personal machines. The cost of memory, secondary memory costs in particular, are viewed as the primary problem area in the development of inexpensive personal machines. The development of a solid state secondary memory device, the so-called "silicon disk," would put personal machines on an equal footing with large shared computer systems. The two candidates receiving most attention are bubble memories and charge-coupled devices (CCDs). Secondary storage devices based on these technologies will be a breakthrough which may signal the

start of major changes in the predominant size of computer systems.

These examples of emerging personal computer systems suggest that a situation in which each user has a dedicated machine of significant power is currently on the edge of economic viability and will certainly be feasible in the near future.

4.5.5 Personal Computer System Design Constraints

When faced with designing an operating system for a new computer, it is important to determine whether current techniques apply to the new environment. For example, would the file system, virtual memory and input/output interface provided by systems like Multics or TENEX serve the needs of a personal computer user? There are several factors that constrain the design of a personal computer operating system in ways that prevent following the exact models set by such systems. First, while physical processors and secondary storage will be dedicated to each user, the power and size of these resources will be less than those for large multiplexed machines. For example, since the storage facilities are not multiplexed among groups of users, the demands for "per user" file storage will be greater in a personal computer than in a shared machine. Thus there are likely to be instances where the needs of the user can not be

directly provided by the personal computer and must therefore be provided by external resources. This is, in fact, the factor that brings us to consider personal computers and network operating systems in the same study. The goal is to determine the nature of services that must be provided by an NOS to augment the capabilities of a personal machine.

A second constraint is the desire to allow autonomous operation of a personal computer to the extent possible. One consequence of this autonomy is that there will be periods of time when a personal computer will not be accessible from the communications network either because it is running in a stand-alone mode or because it has been powered down. This places restrictions on how closely operations between personal computers can be coupled. For example, one of the anticipated uses of a personal computer will be for handling messages between users. Any system that is developed for transporting messages from one personal computer to another must be prepared to deal with the situation where the recipient personal computer is not accessible. Another example concerns the sharing of files between two users. File sharing schemes cannot rely on the constant availability of a particular personal computer. The implementation of system services for personal computers should attempt to minimize tightly coupled interactions with other computer systems.

The effects of these two constraints will appear in subtle ways in the design of the operating system for the personal computer. In most cases there are the reasons for departing from traditional multiplexed operating system designs. In addition, since mechanisms for protecting one user from another are not required in a personal computer system, there will be some simplification of traditional multiplexed operating system functionality associated with interuser protection. However, protecting the user from his own mistakes is still a significant issue.

4.5.6 The Use of Files and Processes on a Personal Computer

This section describes the desired capabilities of a personal computer and motivates some problems an NOS that supports personal computers must solve. To do so, we consider how personal computers would be used and some mechanisms for supporting their use. We assume that the user is trying to solve problems of the sort that concern a typical office worker: accessing information sources, preparing reports and exchanging messages with other individuals. We believe that a system aiding him should be extendible so that it is possible for new programs to be easily installed into it.

Let us consider a possible configuration consisting of a collection of interconnected personal computers and larger machines that augment the capabilities of the personal computers. Figure 17 illustrates such a configuration. Further justification for this configuration is presented below. The resources of the service machines are managed by an augmented version of ELAN. Each user is assumed to have access to a personal computer, perhaps by selecting one machine from a pool and inserting a privately held disk pack. The disk pack will hold the user's personal files, an image of the operating system, user profile information, and commonly executed programs like text preparation programs, interpreters, compilers and message handling systems. The number, size and nature of the commonly used programs will depend on the the personal computer configuration and the available storage.

Files for storing data or programs are important resources in most computer systems. A single user of a conventional shared computer system accesses several different classes of shared files. First, there are unshared files that belong strictly to the user. Examples include profiles for application programs, files that hold messages for the user, partially completed work not yet ready for public distribution, and any files that contain sensitive information. Second, there are files that the user

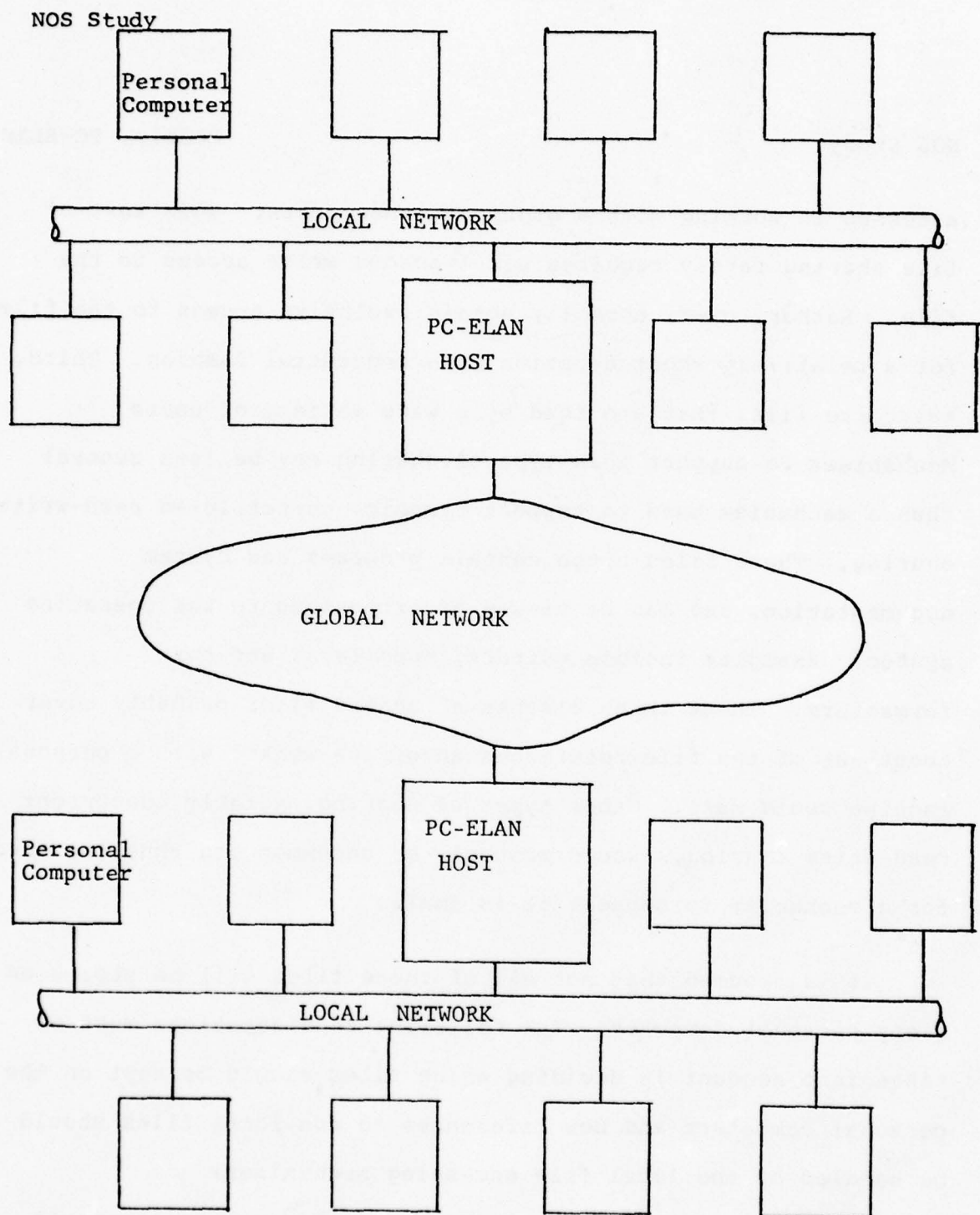


Figure 17.

Configuration for PC-ELAN NOS. Communication within a cluster of personal computers is supported by a local area network. The large ELAN hosts and clusters of personal computers are interconnected by a global network.

accesses in working with a group of other users. This sort of file sharing rarely requires simultaneous write access to the file. Rather, users normally obtain exclusive access to the file for a relatively short duration in a sequential fashion. Third, there are files that are read by a wide variety of users. Mechanisms to support this type of sharing may be less general than a mechanism used to support dynamic, unrestricted read-write sharing. These files often contain programs and system documentation, and can be viewed as extensions to the operating system. Examples include editors, compilers, and text formatters. These three classes of shared files probably cover about 90% of the file references an office worker with a personal machine would make. Other types of sharing, notably concurrent read-write sharing,, would probably be uncommon and thus the need for a mechanism to support it is small.

It is assumed that not all of these files will be stored on every personal computer. The following considerations must be taken into account in deciding which files should be kept on the personal computers and how references to non-local files should be handled by the local file accessing mechanisms:

- The storage facilities of the personal computer are small compared with the storage facilities of the service machines. The problem of limited file storage is compounded by the need to store commonly accessible application programs. In a shared computer system, users divide the

storage costs of these programs, but in a personal computer system each user will generally store copies of the public programs in his own machine.

- One of the main purposes for having a personal computer is to provide uniform, fast response. Any file operations that would result in long delays might be better performed as background tasks.
- When several users modify the same file (perhaps separated in time) then the new contents of the file must be available to each user. Because of the autonomy of personal computers, situations where one user wishes to run in a stand-alone mode while the other wishes to access the shared file must be handled.
- At any instant, the group of files that is likely to be accessed by a user is a small subset of the files that could be accessed. As time progresses, this subset may change, and thus there must be a provision for moving files between the personal computer and a service host file system.

Given the potentially conflicting demands for space in the personal computer's file system, a balance in its use must be achieved. To achieve this balance, space must be allocated for the following types of files:

- Personal files of the user that have no other permanent position in any storage system accessible by the personal computer.

The user has complete (physical) control over these files. For purposes of reliability, it would be possible, although not mandatory, to selectively backup these files on a service storage facility.

- Images of files which are currently of concern to the user and that are stored somewhere else in the service file system.

The relationship between the local image and the image stored in the service file system should be weak -- local

modifications should appear in the service file system only when the local copy is sent back and overlayed on the service copy. The length of time a user might be "currently concerned" with a file could range from minutes to days. During that period, the file at the service storage facility ought to be accessible to other users (as access controls permit); however, files for which images are "on-loan" to users at personal computers ought to be tagged with information that would help settle potential conflicts in their use. The use of files in this fashion is similar in concept to the way files are moved to a tool workspace in NSW and subsequently returned to the NSW file system by a "deliver" operation.

- Files that contain programs used on the personal computer.

These are the "tools" used to access data bases, produce reports and exchange messages.

For the personal computer system to be useful, the size of the local storage system must be large enough to store a balanced mix of these three types of files. Dominance by one type at the expense of others will result in frequently incurred delays while files are shipped to and from the personal computer system. Such "thrashing" behavior could severely limit the responsiveness of the system. Even when file movement occurs in the background, if the balance is bad or the personal computer file system too small then there may be frequent occasions where the user must wait for the completion of file transfer operations.

Processes are important resources of the personal computer system. The initial discussion of the file system has indicated a need for multiple processes within the personal computer:

while time consuming file operations go on in the background (one process), it is desirable to allow the user to proceed in the foreground (another process) with tasks than can be performed before the file operation is completed. Before becoming committed to this requirement, let us consider what seems at first to be a simpler system structure: a single-process system. The following discussion argues that a single-user multi-process system is not inherently more complex than a single-process system that handles all of the requirements of complex hardware devices.

A single process machine executes one logical sequence of instructions. Different tasks are performed when the process executes instructions in different areas of the process's address space. While the contents of the address space may be changed by use of overlays, in theory there is only one logical operation in progress at a time. Timing requirements of the hardware and the goals of the human interface force changes in this simplistic view. When these are taken into consideration, the notion of a small amount of processing, the "interrupt", arises. This is significantly different from the processing being done by the single process because:

- There maybe timing dependencies associated with servicing an interrupt.

Quite often a certain action must be performed within a short time interval or data will be lost.

- There may be more than one source of interrupts.

The problem is that an interrupt routine cannot generally assume that it will run uninterrupted.

- An interrupt routine may access a significant amount of private data.

Giving the interrupt routine access to this private data typically involves providing pre-allocated areas of main memory that are thus taken away from the user's process address space.

Modern operating system designs avoid the use of interrupts as much as possible. The sophisticated programming techniques needed to properly service interrupts make systems more complex and difficult to maintain and modify. For example, as a result of critical timing requirements, a different subroutine calling convention is often used for interrupt calls than for regular calls. The interrupt calling sequence is typically much faster, at the expense of not setting up a complete execution environment for the interrupt routine. This leads to two styles of programming. The style used for interrupt routines has to cope with a weaker execution environment than the style used with regular routines.

There are two related differences between a single-process system with interrupts and a multi-process system where interrupts are converted into process activations: the extent of the state change when the processor is switched between tasks and the speed of performing preemptive process switching. Generally in a single process system where interrupts force the processor to execute interrupt service routines, a minimal amount of state information is changed when the interrupt routine is given the processor. In addition, the interrupt routine preempts the running process and executes immediately. Frequently, one of the initial tasks of the interrupt routine is to save more of the state left behind from the interrupted process. Because the state change is minimal, switching the processor from the running process to the interrupt service routine is typically done by the hardware.

In a multiprocess system, interrupts are usually converted to process wakeups. Starting an interrupt routine is equivalent to starting any other process in the system. This implies that any time an interrupt occurs, a preemptive scheduling decision must be made and, if the interrupt service process has higher priority than the currently running process, a full process state switch performed. The success of this approach is largely determined by how quickly preemptive scheduling decisions and

process switches can be made. Currently most of these mechanisms are implemented in software. Since preemptive scheduling and process state switching are reasonably well understood, it is probably feasible to implement these functions in hardware so that the timing restrictions of devices that generate interrupts can be met.

The conclusion from this analysis that the two approaches to providing multiple instruction sequences are actually quite similar. The essential difference between the two views is the extent to which hardware mechanisms are employed in process switching tasks. An operating system for a personal computer that provides multiple processes is not necessarily more complex than one that provides a single process with the ability to run interrupt service routines. The ability to develop a simple multiprocess mechanism for a personal computer will be aided by two factors. First, advantage can be taken of the cooperative nature of processes in the personal computer. This reduces the need to provide absolute information security between the processes. Consequently, the process switching mechanism can probably be an abbreviated version a fully secure process switching mechanism. Second, to ensure fast process switching, care must be taken during system design when the details of process state information are fixed. There is a tradeoff between

mechanisms that make the programming interface cleaner and the amount of additional process state information that must be maintained and, therefore, switched at each process change. A balance must be achieved so that the need to develop two different styles of programming (related to fast and slow process switching) does not arise.

There are numerous uses of processes possible on a personal computer. To prevent the responsiveness of the personal computer from degrading, the existence of processes not performing tasks directly requested by the user should be rigidly controlled. This means, for example, that, in general, a user should not be able to create processes on another user's personal computer, and that interprocess communication with processes outside of the personal computer should be controllable so that unwanted messages can be refused inexpensively. However, applications should be able to use processes in a flexible manner. To help compensate for the delays associated with network transactions a user should be able to invoke background file transfers while he proceeds with other tasks. The notion of parallel tasks, of which background file transfer is an example, should be generalized to apply to a variety of potentially parallel tasks. The user should be able to start tasks that require little interaction and have them continue while he directs his attention to other tasks that can proceed in parallel.

While the intended use of these small machines is for personal use, user authentication is still an issue. It should be possible to leave the computer unattended with the assurance that someone else cannot walk up and gain unauthorized access to personal data. In the NOS context, when the personal machine draws on the resources of a central service computer, requests must also be authenticated. For example, if a user wishes to copy a file from a service machine to his personal computer's file system, his identity must be passed along with the request for purposes of access control. From the standpoint of the service system, the personal computer cannot be trusted to present the user's identity without authentication because there is no assurance that it will do this correctly. As a result, authentication of the user at the service machine must occur. This second authentication can be automated so that the user does not have to be directly involved: for example, keys or passwords required for authentication at the service machine can be stored on the user's disk (perhaps encrypted as a key used for access to the personal computer). When secondary authentications are required, the personal computer can automatically decrypt the key and provide it along with the resource request. This is similar to the approach to interhost authentication taken by the RSEXEC system.

At this point, a rough characterization of the attributes of a personal computer system has been presented. Below, the requirements associated with interconnecting personal computers together and to service machines will be discussed.

4.5.7 Integration of ELAN and Personal Computers

The integration of resources managed by ELAN with the resources of a personal computer system requires a choice between alternative viewpoints. One view is that the user's PC process has the same requirements as a process on one of the larger ELAN hosts. In particular, convenient access to non-local resources (files and processes) is as desirable for a user of a PC as it is for the user of an application process of ELAN. From this perspective, an interaction between a personal computer and ELAN is the same as the interaction between two ELAN hosts. A contrasting view is that the user of a PC performs significantly different types of operations on the PC than on the larger ELAN hosts. The PC might be used for highly interactive operations while the ELAN hosts would perform less interactive, batch oriented operations. Within this framework, file transfer and process interaction between the PC and the ELAN host could be done by explicit user commands that made the remote interaction completely visible. This is similar to the approach taken in ATF.

The main difference between these two views is the extent to which non-local operations (e.g. file movement, process creation) are performed automatically for the user. With the former view, most non-local interactions are handled automatically while with the latter view, the user must manually guide the system in all non-local interactions. Disregarding issues of efficiency and performance, fully automatic operation seems desirable. The user is free from concern with the details of accessing non-local resources. When efficiency is a factor, there is a tradeoff between the amount of automatic operation and the amount of overhead associated with it -- overhead in the form of CPU cycles and perhaps more importantly, dedicated address and storage space.

With the first view, a considerable amount of background processing must be done on the PC to fully integrate it into ELAN. This means that at least some of its processing power will be diverted from interactive user operations to handling background tasks of the type required of an ELAN host. For the partially integrated PC, no background processing is required and thus all of the cycles of the PC can be dedicated to the interactive needs of the user. Background processing requires that some memory resources be dedicated to it. For the fully integrated PC, this could amount to a significant overhead. For

a manually integrated PC, the memory resource is required for intermachine operation only when non-local interaction is invoked by the user.

There are several other approaches to integration between these two extremes that seem reasonable. Throughout consideration of these alternatives two guidelines should be observed. The first has been discussed previously: tasks not explicitly satisfying requests made by the PC user should not be performed on the PC. The second guideline is based on a desire to exploit what appears to be the inherent data security present in a PC. It should be possible to physically prevent a data file on a PC from being accessed by disconnecting the machine from outside communication or by removing the storage media holding the file from the PC. The potential for the user to control the accessibility of data on a PC seems to offer a level of data security unavailable on large multiplexed machines. To exploit this potential for security it must be possible at any time to separate one or more of the PCs from the rest of the PCs and the ELAN hosts. The capability of the PC to operate autonomously is desirable not only from the standpoint of robustness, but also from that of data security.

The following are several approaches to the PC-ELAN integration. They are presented in order of increasing integration.

- The PC acts like a terminal command language interpreter to ELAN. Any interaction between the PC and ELAN must be expressed in the ELAN protocol used between terminal and home site CLIs.
- ELAN is modified to allow file transfers from the support computer to the PC. A program is run on the PC to perform the transfer. The major change to ELAN is to allow a remote, non-ELAN Kernel Process to be the recipient of a copy of a file stored on an ELAN host. The notion of a copy of a file is new to ELAN. It is an idea, however, that is much less difficult to design and implement than the existing ELAN support for dynamic reference to remote files.
- ELAN is modified to perform several tasks for the PC that were not required previously to support ELAN command language interpreters. These tasks may be characterized as mechanisms to couple PC actions with the resources on ELAN. Examples include adding attributes to ELAN files to reflect their use (in the form of an outstanding copy) by the PC, and fielding interprocess communication messages directed to a process on a PC when the PC is disconnected from the network.
- The PC and ELAN are modified so that ELAN resources are integrated into the set of objects that may be directly referenced by processes running on the PC. This is designed to be one-way cooperation. An application process running on the PC may access resources of ELAN, but the PC resources are not available to processes running on the ELAN host. This provides some uniformity in accessing resources on the PC and ELAN, mainly in the area of naming. However, the possible modes of access to files is different depending on the location of the file with respect to the PC application program. Files on ELAN are never opened for direct access by a PC based application program; rather, the contents of files are copied to or from the PC file space. This is primarily a consequence of the desire to allow the PCs to operate autonomously. If the PC is completely under the control of the user, then it cannot be relied upon to engage

in complex file sharing protocols. Instead, copies of files are transferred to the PC and it is the responsibility of the PC to transfer modified copies back to ELAN hosts. If the PC fails to send back the file, then the only user affected is the user of the PC -- the original contents of the file are preserved. ELAN Application Processes (APs) are integrated into the accessing environment of the PC in a manner similar to the way ELAN files were made available to application programs running on the PC: the use of ELAN APs by PC application programs is permitted but the reverse is not. A PC application program may create processes in the ELAN process hierarchy but ELAN APs may not create processes on the PC.

- There are several further steps along the spectrum of possibilities for integration until the PCs are essentially additional ELAN hosts. The major change is to allow all PC resources to be accessed by arbitrary APs in ELAN. While in general this is contrary to the initial goal of restricting tasks on the PC to those directed by the user of the PC, there are instances where access by ELAN to PC resources is desirable and in fact does not violate this principle. These are cases where the user of the PC has created an ELAN-AP to perform some task and the ELAN-AP then needs direct access to resources on the PC (like a file) to accomplish the task. It is a very short step from providing mechanisms that allow this form of access to mechanisms that allow a PC to be a full ELAN host.

Each of these alternatives outlines a different level of integration for a PC-NOS combination. In addition, going from the first point to the last provides a phased implementation approach -- each view of integration includes those views above it in the list. The integration of PCs with service machines was expressed with the assumption that ELAN would act as the operating system for the machines that provided service. Other, less sophisticated operating systems could also serve this

purpose. However the requirements of the PCs may very well require capabilities similar to those provided by ELAN.

If PC files are to be accessed by ELAN APs, then there must be a way for ELAN-APs to reference the files by name. The most consistent way to do this is to add them in some fashion to the ELAN file hierarchy. ELAN-APs can then reference files by a pathname regardless of whether they are stored on one of the ELAN constituent hosts or on one of the PCs. Two possible approaches for determining the position of PC files in the ELAN file hierarchy are apparent. In the first, files on the PC can be treated just as files are treated in the ELAN hierarchy: the position of the file in the hierarchy is determined by its function. For example, program source files that implement a compiler would be kept in a directory devoted to the compiler rather than the directories of the individuals that created the separate program modules that make up the compiler. Under this view, when the files of a PC are integrated into the ELAN hierarchy, a catalog entry would be added to an ELAN directory with an indication that the file contents resided on a PC. The files of a given PC would be incorporated into several directories in ELAN depending on their function.

An alternative view is that the files in the PC are totally different from files in the ELAN hierarchy. The PC is often used as a work station with many files residing on the PC only for relatively short periods of time. There really is no long-term association between a particular file on the PC and the the files of ELAN that belong to its functional group. Rather, a stronger association exists between a file on the PC and the user of the PC. From this perspective, any references to the file in the context of the ELAN should be made through a point in the storage hierarchy associated with the user of the PC. With the second approach, the amount of new information that has to be introduced into the ELAN storage hierarchy is *minimized*. This latter approach appears to be the most attractive one.

4.5.8 Communication Requirements of Personal Computers.

Shifts in the predominant mode of computer system support away from large time-shared computers toward smaller personal computers will not diminish the need for user-to-user or process-to-process communication. To support this communication, personal machines and support computers must be connected together. Networks such as the ARPANET represent a cost effective means for the interconnection of the larger multiplexed machines that are predominant today. With 10 to 100 users per

host, the cost of a connecting a user to the ARPANET is from \$200 to \$2000 (assuming that an IMP costing \$80,000 can support four hosts). If there is a host for each user, the economies of scale of the ARPANET are no longer valid. Instead, some different, lower cost scheme must be developed.

Table 2 describes the communication needs of three different types of interactions that are likely to occur in a group of PCs and supporting PC-ELAN constituent hosts. The first is communication between PCs that are situated close together geographically. An example of this type of communication is the use of interprocess messages between parts of an application running on several different PCs. The two characteristics of communication traffic that determine the type of support needed by an underlying transmission medium are delay requirements and message size. To ensure the general responsiveness of applications running on a PC the delay in transmitting a message from one PC to another should be low. The message size for such applications is typically small.

A second type of communication that must occur is between PCs and PC-ELAN support hosts. We assume that one of the support hosts is located fairly close to a group of PCs (e.g. in the same building). An example of this class of communication is file

TABLE 2

Patterns of Communications in PC-ELAN

| Type | Example | Importance of Low Delay | Size of Messages | Geographic Distance |
|-----------------------|----------------------------|----------------------------|---------------------|------------------------|
| PC to PC | Application Interaction | Great | Small | Small |
| PC to Support | File Retrieval, Paging | Great | Large | Small |
| Support to Support | File Retrieval | Desirable | Small and Large | Large |

retrieval from the PC-ELAN host. Unlike the PC-to-PC application interactions, PC to PC-ELAN host communications will involve large amounts of data. Here high throughput as well as low delay is important if the support host is to be considered an effective extension of the PC.

Finally, there are the communication requirements for the PC-ELAN hosts. The message size here will be mixed. There will be short messages between the KPs to perform control operations and longer messages or larger transfers of data to support file movement. The overriding characteristic of this class of communications is the potentially large distances between constituent hosts. Like the other types of interaction, low delay and high throughput are desirable attributes. However, there is reason to believe that because of the physical distances, communication between PC-ELAN constituent hosts will not be as fast as between PCs that are physically close together.

The large number of hosts that must be interconnected could result in congestion if all of the messages exchanged between hosts had to travel on the same network. There is, however, a natural subdivision of hosts in such a collection. First, the PC hosts and the PC-ELAN hosts have somewhat different intercommunication requirements. Second, geographically

centralized organizations form subsets of PCs that are more likely to communicate with each other than with PCs outside of their group.

As a result of these considerations, a configuration of clustered PCs connected together and to a PC-ELAN host by a high speed local area network seems appropriate. In addition, the set of PC-ELAN hosts (perhaps one per PC cluster) is also connected together by a global network such as the ARPANET. Figure 17 illustrates this interconnection. Each PC-ELAN host acts as a support machine for the cluster of PCs to which it is directly connected by a local area network and as a "gateway" to other PC clusters and PC-ELAN hosts. Local area networks provide the high speed and low interconnection cost required by the PCs, and the global network satisfies the communication requirements of the PC-ELAN hosts. The state of the art in local networks is discussed in the Appendix.

4.5.9 PC-ELAN as an NOS

Many of the comments regarding ELAN as an NOS apply to PC-ELAN. The primary improvement of PC-ELAN over ELAN is the ability to utilize advances in electronics that will lower the price of hardware to the point where dedication of small machines

to individual users will be possible. Building on the unified model of resources developed in ELAN, PC-ELAN represents an approach for allowing small machines to access an expanded resource environment.

The main emphasis of the discussion on personal computers was to preserve the dedicated aspect of these machines. A personal computer should be responsive to the interactions of a user. Unlike current time-shared computer systems, the load on the system should not be influenced by the work of other users. A person should be able to disconnect his personal computer from outside communication at will. This isolation can be used to provide data security. The support offered by PC-ELAN provides a common area for information sharing between users of personal computers as well as a place for storing files that will not fit in the limited long term storage facilities of the personal computer.

There are still many issues to resolve in making effective use of personal computers. Once several applications utilizing personal computers as the underlying hardware base have been developed, requirements for support systems that would otherwise be difficult to anticipate will become apparent. Such requirements will serve as the basis for future network operating systems.

5. Comparison of NOS Models

As the five NOS models were described in Section 4 comparisons were made among them in order to explain various system design decisions and features. In this section we further compare the systems in a variety of areas.

For the comparisons to be meaningful, care must be taken in making them since the systems modeled are substantially different. ATF, ELAN and PC-ELAN are partial system designs resulting from this study, whereas RSEEXEC and NSW are implemented NOS systems. Furthermore, neither RSEEXEC nor NSW can be regarded as a finished product. Although implementation efforts on RSEEXEC ended over three years ago, many interesting ideas remain that could be explored in the RSEEXEC context. Implementation of the NSW is underway currently, and it will be some time before it is considered complete.

Therefore, a point by point direct comparison of the five systems will not be attempted. Instead, we shall compare some consequences of the different design philosophies represented by each system. These consequences are unlikely to change appreciably even with implementation effort in the case of the three study models or with additional implementation effort in the case of the two real systems. Thus, they should remain valid throughout the system's existence. It is our feeling that examining implications of the design philosophies will provide additional insight into the concepts embodied by each system. By

focussing on the differences between the systems, we hope to illustrate the great flexibility open to the designer of an NOS.

Another point that should be made here is that the ATF system is not a "total" or integrated NOS, whereas the other four systems represent different approaches to realizing total NOS systems. Because many of the points of comparison below relate to aspects of the systems as total systems, there is often not much to be said about ATF systems.

5.1 Role of NOS Administration

For ATF systems and for RSEXEC there is no concept of a NOS administrative organization. Both systems act to facilitate user access to network resources but neither provides support for system wide administrative control functions. As a result, the constituent hosts in these systems remain totally autonomous in an administrative sense. Each user must make separate arrangements with the administrative organizations for each of the hosts whose resources he wishes to use. Once these arrangements have been made and the appropriate user profiles have been initialized, the user can make use of the NOS features supported by ATF or RSEXEC. However, the user must still deal with the various accounting and billing procedures of the different host organizations.

To provide administrative services, such as the ability to accurately account for resource usage, technical support for

administrative services must be implemented by the NOS. In addition, there must be an organization responsible for performing routine administrative tasks, such as establishing and managing user accounts as well as defining various system policies. NSW, ELAN and PC-ELAN each provide the technical support required for the operation of an NOS administrative organization.

NSW supports the concept of a single NOS administrative organization which users can interact with to establish NSW accounts. Because accounting and billing is handled by the NSW administration, the user has a single NSW account and a single procedure for paying it. Once an NSW account has been established for a user, his ability to use any of the resources managed by NSW, regardless of their host location, is limited only by any access controls set by the system administration or his project manager. No additional accounts need be established.

NSW is designed to allow system and project administrators to exert strong control over the ways the system is used by different users. For example, system administrators control which software packages are included in the NSW tool set and which subsets of the tools are available to various user groups. A project manager can, within his own project, control which tools the various project members can use and, in addition, the amount the various tools may be used by project members.

At present, it is unclear whether there will be a single DoD-wide instance of NSW to support all DoD NSW usage or many independently operating instances of NSW systems, each to support a collection of related DoD software production projects. Regardless of whether there will be one, a few, or many instances of NSW systems, the above remarks remain true. If there are multiple NSWs, each will be administered by a single organization.

Like NSW, ELAN/PC-ELAN is designed to support an NOS administration. The role of the NOS for an ELAN/PC-ELAN system is somewhat different than that for NSW. For ELAN/PC-ELAN the NOS administration exists principally to provide a single convenient contact point for the user in establishing an NOS account and to permit uniform NOS accounting and billing. Because ELAN/PC-ELAN are designed to be general purpose systems, there is little motivation to administratively control the ways the system is used to the extent done within the NSW software production environment.

ELAN/PC-ELAN would be more amenable to a "broker" or "middle man" approach to NOS administration. The broker would buy wholesale from the resource bearing hosts services which he would retail to the users. The ELAN/PC-ELAN systems would provide the mechanisms for doing this. The user would see the uniform accounting and billing procedures of the broker and would have relatively unconstrained use of the resources provided by the NOS.

5.2 NOS Resource Management - Centralized vs Decentralized.

The four "total" NOS systems exhibit somewhat different approaches to resource management and allocation. In NSW the resource management function is centralized whereas in RSEXC and ELAN/PC-ELAN decentralized approaches are used.

For purposes of discussion we identify three points of interest on the centralized/decentralized spectrum: logically and physically centralized; logically centralized and physically decentralized; logically and physically decentralized. The resource management function is said to be logically and physically centralized if there is a single module on a single host that has responsibility for it. It is logically centralized and physically decentralized if there is a single functional module supported on a number of hosts that has responsibility for it. Resource management functions for an NOS are said to be logically and physically decentralized when the responsibility for it is distributed among the constituent hosts.

Decentralization is largely a question of degree. In fact, the differences between the last two points on the centralized/decentralized spectrum are somewhat subtle. One point of difference between the two concerns the range of resources controlled by the resource management modules. In a logically centralized and physically decentralized scheme every module (potentially) controls every resource, whereas in a logically decentralized scheme different sets of resources may be

managed by different modules. Further differences are perhaps best explained by example.

In NSW the resource management is implemented by the Works Manager component. For the current implementation the Works Manager is supported by a single host. Thus, in the NSW, resource management and allocation is logically and physically centralized. A design for a distributed NSW Works Manager has been developed. In this design the Works Manager function would be implemented by identical Works Manager modules running on a subset of the NSW hosts. The distributed NSW Works Manager represents a logically centralized and physically distributed approach to NOS resource management. It is logically centralized in that a Works Manager host, which in general is different from both the host attempting to access a resource and the host supporting the resource, must be consulted on each attempt to access an NSW resource; it is physically distributed in that the Works Manager function is supported by a number of different NSW hosts. For RSEXEC resource management is logically and physically decentralized in the sense that a user's ability to access resources is determined only by the accessing host (that is, the host RSEXEC is running on) and the host or hosts that support the resources. Resource allocation and management in ELAN/PC-ELAN is logically and physically decentralized in the same sense. There is no host or subset of hosts responsible for resource management; rather, the responsibility is distributed among all of the hosts.

The approach to resource management chosen for an NOS influences a number of system characteristics:

- System Reliability.

Systems for which the resource management function is logically and physically centralized are vulnerable to failures of the single resource management host. When that host fails system operation (or at least those operations requiring resource management decisions) must be suspended even though other hosts may be operational. A system employing logically centralized but physically decentralized resource management is not vulnerable to single host failures. Its reliability can be made arbitrarily high by increasing the number of resource managing hosts. However, it remains vulnerable to multiple host failures since the possibility remains that all resource managing hosts will fail. In such cases system operation requiring resource management decisions must cease even if the hosts accessing and bearing the resources are fully operational. Systems using logically and physically decentralized approaches are potentially the most reliable systems. For them, system operations can continue regardless of the number of host failures for configurations in which the accessing and resource bearing hosts are operational.

- Host Autonomy.

Autonomy is related to system reliability. For systems in which resource management is centralized, either physically or logically, the ability of resource bearing hosts to function as part of the NOS is impaired if they are unable to interact with a resource controlling host. This is the case for the NSW. For more decentralized schemes, where the resource control function is distributed among the resource bearing hosts, autonomous NOS activity is possible in the event a resource bearing host is partitioned from the rest of the NOS. Of course in this case only resources resident on the isolated resource bearing host could be used, but the mode of operation from a user's point of view is identical to that for a fully configured NOS.

- System Performance.

For a physically centralized scheme the resource controlling host may become a performance bottleneck since it must mediate every attempt to access a resource. In physically decentralized schemes the load of performing resource allocation can be distributed among several hosts thereby

significantly reducing or eliminating this potential performance bottleneck. Schemes employing logical centralization of resource management potentially require more interhost interactions for each attempt to access a resource than schemes that are logically decentralized. This is the case because, in general, the resource accessing, resource bearing and resource managing hosts may be different. Consider, for example, the situation in which a resource being accessed resides on the same host as the accessing process. A logically centralized scheme requires an interhost interaction with the resource managing host to complete the access whereas no such interaction is needed for a logically decentralized scheme. Thus, for applications that demonstrate locality of reference systems employing logically decentralized resource management are likely to perform better.

- Implementation Complexity.

Physically centralized schemes are the easiest to implement. Decentralized schemes require coordination among the distributed parts to accomplish the mutual exclusion and synchronization required to ensure consistent operation. For example, the distributed Works Manager design includes a synchronization mechanism to guarantee that all copies of the WM data base remain consistent with one another. In ELAN, the file catalogue hierarchy has a two tiered organization which requires interhost synchronization actions only for operations performed on the upper tier; operations on the lower tier which are expected to be more frequent require no interhost synchronization.

5.3 Visibility of Distribution

An important issue to resolve in the design of a network operating system is the nature of the interface to the network resources as seen by a user or a program. Part of this issue is the degree to which the distributed nature of the underlying system is visible. One extreme position is to provide complete invisibility, where a user is not necessarily aware of the network operations being performed on his behalf, nor is he able to exercise direct control over the use of distributed resources.

At the other extreme is complete network visibility, where the user is aware of the network and its constituent hosts, and must directly assert control over the selection of resources to service his requests. The system models developed in this study span the design spectrum from both extremes.

ATF systems require users to perform their own host and resource selection. An ATF user specifies where programs are to be run, which hosts are to store files, and explicitly transfers files from one host to another. As noted in Section 4.1, for a sophisticated user willing and able to deal with the hosts at this level an ATF system can provide a convenient and very cost effective means for using network and host resources.

The NSW attempts to mask almost all details of network operation and system distribution from the user, and thus tends to the invisible end of the design spectrum. The NSW design is based on the belief that dealing with the network or the constituent host systems at any level would impair a user's ability to easily utilize combinations of software development tools. The NSW system itself currently assumes all responsibility for selection and placement of system resources, without notifying or consulting the user. Neither the syntax nor the semantics associated with the NSW user interface includes any provision to specify actions directly relating to the distributed

AD-A056 078

BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS
NETWORK OPERATING SYSTEMS.(U)

F/G 9/2

MAY 78 R H THOMAS, R E SCHANTZ, H C FORSDICK
RADC-TR-78-117

F30602-76-C-0425

UNCLASSIFIED

NL

4 OF 4
ADA
056078



END
DATE
FILMED

8-78

DDC

nature of the system (1). As a result, all network resources are uniformly and easily accessible within a simple user framework.

There are a number of factors which suggest that a more "visible" approach to network systems is sometimes more appropriate. In currently available systems there is frequently an appreciable decrease in performance when remote resources are accessed. Also, various similar system resources may have slightly different characteristics on different hosts or host types. For example, a manuscript preparation system supported on two different host types, although written by the same programmer to the same specifications may behave slightly differently on the two hosts. The "Multics Runoff" package which runs on Multics, TENEX, and TOPS-20 hosts in the ARPANET is a case in point. In certain applications, such as those occurring in distributed data management, a user or his program might want to exploit knowledge of the location of various data items in order to achieve more responsive or more reliable service. Finally, certain operations may be supported only for adjacent entities. In this case distribution and the relative location of resources must be visible.

-
1. However, network considerations intrude somewhat into the user's conceptual model of the system in such areas as the movement of files between NSW file space and the secondary storage not under NSW control, and also with the use of tool workspaces. The visibility in these areas represents the necessity of dealing with operational realities rather than a part of the system design philosophy.

For these and other reasons, some system designers have opted for user and program interfaces which acknowledge the distributed nature of the system, and allow users to influence, if not specify, resource selection patterns. Coupled with this is usually a set of judiciously selected system defaults chosen to minimize the requirement for user involvement in resource selection decisions.

Both RSEXEC and ELAN are visibly distributed. With RSEXEC users deal with the distribution in terms of physical host names. For example, a full RSEXEC file pathname includes a host name component which specifies the host where the file is stored. In ELAN distribution is dealt with in terms of logical relationships between resources, such as adjacency and remoteness. The two tier file system hierarchy of ELAN provides the means for transforming between file pathnames and physical host locations. Both RSEXEC and ELAN also provide mechanisms for operating in the NSW-like fashion where a user is not required to specify details relating to distribution.

For PC-ELAN visibility of the distribution seems inevitable given the system architecture of personal computers supported by an ELAN-like NOS. The user is well aware of the boundary between his personal machine and the rest of the system. The thrust of PC-ELAN is to make it easy for the user to obtain support from the larger hosts rather than to hide their existence.

5.4 Extendible Program Set

Although the designs for RSEXEC, NSW, and ELAN/PC-ELAN all provide means for users to execute programs on the network hosts, there are design and philosophic differences between the NSW on one hand, and the RSEXEC and ELAN/PC-ELAN systems on the other. For RSEXEC and ELAN/PC-ELAN, the object that can be placed into execution is obtained from the distributed file system. To add to his collection of executable programs, a user need do nothing more than create a new file. In NSW, the object that a user can place into execution can be selected only from a system wide tool list maintained separately from the NSW file system by the Works Manager component. New programs can be added to the available tool set only by action of the NSW administrative staff. Although at first this might seem arbitrary, it is a result of the NSW philosophy to provide access only to well debugged and documented programs, and to extend the concept of managing software projects to include the ability to enforce the use of certain programming tools.

The somewhat closed tool set supported by NSW exemplifies another distinguishing characteristic. Namely, NSW is not designed to be the execution facility for completed software, nor should it be viewed as a general purpose computer utility. The software created within NSW is meant to be exported to machines outside of the NSW domain for production runs. RSEXEC to a certain degree and ELAN/PC-ELAN even more, were conceived in more

general terms, and can be viewed as attempts at self contained systems which would ultimately lead to a general purpose computer utility.

5.5 Program Support Services

ATF systems provide little in the way of program support services beyond the ability to start a program running on one of the network hosts. Either the user himself or his program must act to configure the computation so that any data referenced by the program is adjacent to it.

RSEXEC, NSW, and ELAN/PC-ELAN all provide more complete program support services through NOS program execution environments. RSEXEC is the most limited in this regard. As previously noted RSEXEC supports a program execution environment for locally executing programs (1). The RSEXEC program support services are provided through encapsulation only. Because these RSEXEC features were designed principally to extend the single host TENEX/TOPS-20 environment to include the network, the RSEXEC program execution environment does not differ significantly in nature from the single host environment. Consequently, RSEXEC

-
1. While the ATF commands supported by RSEXEC allow a user to start programs running on remote network hosts, the NOS program execution environment is not supported for these programs. Therefore, remote programs execute in the environment provided by the remote operating system. Section 4.2 described how the RSEXEC program execution environment could be extended to support remote program execution.

provides no NOS primitives (such as interhost interprocess communication operations) which programs written specifically for the network environment can use.

The NSW program support features are considerably more sophisticated. Within the limitations discussed in Section 5.4 NSW allows program execution on any of the tool bearing hosts. As discussed in Section 5.3 the user has little control over where a tool (program) executes. NSW provides program support services in two ways. Like RSEXEC, NSW provides an encapsulation interface which allows programs written for a single host environment to execute as NSW tools. NSW encapsulation transforms tool calls upon the single host operating system into equivalent calls upon NSW system operations. The environment provided by NSW, including its file system and other resources, differs in significant ways from the environments of the various tool bearing hosts. Thus, while NSW encapsulation has proven very effective for integrating existing software packages into the system as tools, the transformations it makes can often only approximate the original intent of the tools. For this reason NSW provides a direct interface which allows tools to directly invoke NSW system services. This second, non-encapsulation interface is intended for tools written specifically for the NSW environment and that are designed to take advantage of the unique features provided by NSW.

The intent of the ELAN program support services is to provide programs with a direct interface to the NOS features. However, we believe that an ELAN implementation would have to also provide an encapsulation interface as well. For PC-ELAN the program support services for programs executing on the "large service" hosts would be similar to those provided by ELAN. As discussed in Section 4.6, there are a number of possibilities for the program support services provided by the personal computers in PC-ELAN. The particular services provided would depend on the sophistication of the user, as well as on the applications and their expected usage patterns.

5.6 Interference Between NOS and non-NOS Activity

All five NOS systems are designed to be built upon existing host operating systems. The NOS systems and their users share host resources with normal non-NOS users of the hosts. An individual computer system can be a resource bearing host in an NOS and at the same time directly service non-NOS users. A point of comparison among the NOS models is whether non-NOS access to resources managed by the NOS is permitted. This is a particularly important issue for modifiable resources such as data bases and user files.

ATF systems act merely to facilitate access to the resources controlled by network host systems without attempting to "repackage" these resources. Thus, for an ATF NOS there is no

notion of NOS resources apart from the resources directly accessible to non-ATF users of the constituent host systems.

The RSEXEC hosts permit non-NOS activity on NOS resources. RSEXEC uses local host file systems directly for maintaining its own file system status information. No attempt is made within RSEXEC to maintain a permanent catalogue of user files. A user's composite directory is recreated at the start of, maintained throughout the course of, and discarded at the end of each user session. This approach has the advantage of permitting autonomous operation. An RSEXEC can construct a user's file catalogue (or that part of it that has information about accessible files) directly from the constituent hosts normal file catalogues without interacting with an NOS resource catalogue process which might not always be accessible. It has the disadvantage that RSEXEC files may be manipulated from outside the context of the RSEXEC network operating system. This may lead to occasional inconsistencies which can occur because file activity occurring outside of RSEXEC is not synchronized with the dynamically constructed RSEXEC file catalogues. The impact of such inconsistencies usually is not severe. Typically, an inconsistency manifests itself in the failure of an attempt to obtain a file because the RSEXEC file catalogue entry for it is no longer valid. The RSEXEC itself is resilient to such failures; it merely returns an appropriate error indication to the user or his program.

The user can recover from the failure by instructing RSEXEC to rebuild the portion of his file catalogue that is no longer valid. This reconstruction of the file catalogue could be made automatic. When a file operation fails after having been expected to succeed, RSEXEC could automatically reacquire the appropriate file catalogue information and reinitiate the operation. The current RSEXEC system does not behave in this way; instead it requires the user to initiate his own recovery actions.

An individual computer system can be a tool bearing host within NSW and at the same time directly service non-NSW users. However, the resources supporting the computer as a tool bearing host are dedicated exclusively for this purpose, and cannot ordinarily be manipulated from outside the NSW environment. To accomplish this the NSW Works Manager maintains long term system resource data bases. For example, the Works Manager maintains a catalogue for NSW user files. All file activity within NSW occurs against the NSW file catalogue, and no activity external to the NSW can effect NSW files or the NSW catalogue. All information about NSW files, such as their NSW name, access control information, their location in some host's file hierarchy, and so forth, is maintained in the NSW file catalogue. The file systems of the constituent hosts are used only as repositories for NSW files. This prevents inconsistencies of the sort described above that can occur in the RSEXEC system.

In the design of the ELAN model care was taken to permit autonomy of operation as in RSEXEC but at the same time to prevent inconsistencies that can result from non-NOS access to NOS resources. In our conception of an ELAN system, all constituent host resources would be accessible only through the NOS interface, and all individual host resources would be available through the NOS whenever the individual host was reachable. That is, there is a more thorough integration of the NOS with the individual host in that all host activity would be through its NOS interface. This is made possible, in part, by maintenance of long term file catalogue information in the replicated upper portion of the two tiered ELAN file hierarchy as described in Section 4.4.

5.7 Implementation Considerations

All of the five NOS systems were designed to be implemented upon existing operating systems for the constituent hosts rather than directly upon the host system hardware. Consequently, none perform certain standard functions, such as process scheduling and memory management, typically performed by single host operating systems. Rather they rely on the underlying operating systems to perform these functions. The implementations of these NOS systems are directed toward providing expanded functionality, such as distributed file systems and interhost interprocess communication, required to support users in a network environment.

With the exception of ATF, which can use a single agent implementation approach, all of the models require a distributed agent approach. Because they provide significantly more functionality and require a distributed agent approach, implementation of "total systems", such as RSEXEC, NSW, and ELAN/PC-ELAN, are a much larger undertaking than implementation of an ATF system.

RSEXEC, being a very early NOS, uses in its implementation the user/server paradigm that evolved with the early ARPANET protocols. As described in Section 4.2, there is a user's agent, the RSEXEC program, that implements the user command language and the program execution environment, and there are agents for the resource bearing hosts, the RSSER programs, which cooperate with user agents to provide access to the resources on their hosts.

In NSW and ELAN/PC-ELAN the implementations are partitioned along more functional lines. For NSW there is an identifiable system component for each of the major NOS functions: the Front End component for the user interface; the Works Manager component for resource management; tool bearing host file packages for file translation and movement; and tool bearing host Foremen for tool control. The ELAN/PC-ELAN implementation approach, like that of NSW, exhibits functional modularization. On each host there are kernel processes corresponding to the major functions such as process control (PC-KP) and file management (FM-KP).

6. NOS Implementation Strategies; Requirements for Constituent Host Operating Systems

Certain implementation approaches and mechanisms reoccur in the two NOS models that have actually been implemented and the other three models for which implementation strategies have been considered. This section discusses some issues, problems and techniques that are common to many NOS implementations.

- Service Processes

An effective approach for building an NOS, from the stand point of simplicity and minimizing impact on existing systems, is to implement NOS functions by dedicated service processes. The use of service processes as an implementation strategy is not limited to network operating systems. Many operating systems implement services such as line printer spooling and file system backup, and even lower level functions such as page fetching and processor scheduling, as separate processes. A host which is to take part in an NOS should have provisions for long term service processes. In the models presented here, service processes either implement all of the NOS functions, as in the RSSER service process for RSEXEC, or implement one part of the NOS function, as in NSW and ELAN. In addition to the normal properties of a process, an operating system should support

automatic startup and restart. When an operating system is started, service processes for the NOS functions must also be started. In addition, if an NOS service process encounters an error causing it to be stopped or destroyed, the host operating system must attempt to restart a process for the NOS function, allowing it to clean up any inconsistent data bases and resume service.

- Encapsulation and Direct Call Interfaces to NOS Functions

Application processes that use NOS resources must convey requests to access them to the NOS. Two different approaches for doing so result from different views of the relationship between application processes and the NOS. In the first view, application processes run programs that have no knowledge of the expanded accessibility to resources provided by an NOS. Typically these are programs written without any knowledge of the NOS, either because they were created before the NOS or because they were written for another purpose, and are now being used in the expanded resource environment provided by the NOS. To support these processes an NOS must provide an encapsulation interface where it can examine calls to the host operating system to determine if the application process is actually referencing resources provided by the NOS. To support encapsulation, the

constituent host operating system must provide a mechanism for intercepting calls to the operating system. Both RSEXEC and NSW use encapsulation as their principal programming interface. Extensions have been made to the operating systems of constituent RSEXEC and NSW hosts to allow interception of operating system calls.

An alternative view is that explicit knowledge of the NOS is programmed into the application program. This class of application is programmed to run in the NOS environment and to use the set of primitive operations supplied by the NOS for accessing the network resources. In this case interactions with the network resources can be accomplished by direct calls on the NOS, rather than indirectly by reinterpretation of local host operating system calls.

A direct call interface is preferable to an encapsulation interface in terms of functionality and performance. First, direct calls enable a process to specify precisely the NOS operations desired, whereas for some properties of the local abstract machine there may be no mapping into the NOS operations to achieve the desired effect. Second, even in areas where the functionalities of the NOS and the local operating system are similar there may be a vast difference in the performance

characteristics for the two environments. This could make the tool behave quite differently than originally designed. However, an encapsulation interface is preferable when the reprogramming costs are a limiting factor or when it is required that the application also run in the non-NOS environment. In cases where both types of applications are desirable, the NOS should provide both types of interfaces, allowing the nature of the application dictate the choice of which to use. While RSEXEC provides only an encapsulation interface, NSW provides both thus facilitating the creation of new applications tailored to the NSW environment.

- Communication Between and Within Constituent Hosts

The constituent hosts of an NOS must communicate in order to provide access to distributed resources. The performance of an NOS can, however, become unacceptable if the system relies on excessive interhost communication to accomplish normal tasks. The performance problem with interhost communication is related to the number of separate messages exchanged between hosts rather than the length of the messages, since most messages tend to be short.

As with subroutine calls that pass arguments, interprocess messages have a certain amount of overhead associated with them.

Unless care is taken, the communication overhead can exceed the task being performed in response to a message. For example, each time a pair of messages is exchanged, the sending and receiving processes must be scheduled to run. In systems where scheduling is an expensive operation and where there is no provision for priority scheduling, such interprocess interactions can become a bottleneck. The cost of switching between processes and the delay in being selected as the process to run causes a delay between the time a message is received and the time it can be acted upon. An NOS design that ignores this problem can result in an implementation that requires the exchange of many short messages and which, as a result, performs significantly worse than a conventional single host system. One approach to solving this problem is to consolidate all communication into one message at the beginning of a transaction and a single reply at the end. Another approach, where multiple interactions are inherent to a task, is to speed up the process switching mechanism on the constituent hosts and to provide a priority scheduling scheme where some processes can receive faster scheduling than others. Yet another approach, applicable in certain circumstances, is to avoid interhost communication by supporting local implementations for required services.

- Use of Existing Hardware and Software

All of the systems we have described have been designed with the premise that they are to be built on top of existing hardware and software bases. The advantage of this approach is that most of the common low-level functions performed by an operating system have already been implemented and can be used directly in the NOS implementation. For example, problems of processor multiplexing, long term file storage and device management are already solved by the existing software.

Using existing systems also has its problems. Providing NOS features substantially different from those supported by the underlying operating systems is often difficult. Underlying support that is ill-matched to the requirements of the NOS can result in features that have poor performance characteristics. For example, the interface provided by a constituent host operating system to local resources is often too rich for the needs of an NOS. This richness, while beneficial to a user of the constituent host as a stand alone system, can result in inefficiencies for NOS implementations. Generally an NOS requires that the underlying operating system perform very basic functions for it. Other, higher level operations which make use of these functions are better performed by the NOS. For example,

the only view of the file system provided by the TENEX operating system is one that provides hierarchical structure, partial name recognition, multiple component pathnames, and file version numbers. NSW, which implements its own file system with its own naming and protection schemes, requires much less functionality. Many local host file system features beyond providing a container for data are not needed for NSW. The conclusion is that it is desirable for a constituent host operating system to support several alternative interfaces, at least one of which is a "no frills" interface that permits fairly low level access to resources.

The use of underlying hosts as building blocks for higher level systems can be aided by extendible interfaces to the constituent host operating systems. An extendible interfaces is one where the boundary between the operating system and an application program can be moved to allow the application program to assume some of the duties of the operating system. It is desirable at times to redefine some of the standard operating system functions so that a new resource environment can be provided, but at the same time to preserve other aspects of the environment provided by the standard constituent host operating system. Encapsulation relies on this ability, and efforts to implement an encapsulated environment suffer if there is no way to intercept and redefine standard operating system calls.

- Approach to Reliability

Part of the RSEXEC approach to reliability is to clear a potentially bad system state when an error is detected. This works well because of RSEXEC's general approach of obtaining system data from the constituent hosts rather than keeping it permanently in data bases it maintains. RSEXEC relies on the fact that the underlying hosts are the ultimate sources of correct system state information.

The NSW approach to reliability in this area has to be different from RSEXEC because NSW manages much more state information. Again, however NSW relies partially on the reliability characteristics of the underlying hosts since it stores checkpoints of NSW data bases in the file systems of the constituent hosts. When an error occurs, NSW backs the affected data bases up to a consistent intermediate point and either takes specific remedial action or attempts to proceed from that point.

When reliability is considered in most operating systems, it is from the point of view of making the operation of the system itself reliable. Rarely are reliability mechanisms extended to the application programs that run on the operating system. For conventional systems there is little point in doing so since generally failure behavior is such that the entire system either

functions properly or not at all. Since a distributed system, such as ELAN, can continue to function when some of its components have failed, application programs execute in a failure prone environment where resources they require may disappear and reappear dynamically. Part of the ELAN approach to reliability is to provide application programs with mechanisms that can be used to build reliable services for an environment where the system configuration can change dynamically due to failures. ELAN is somewhat more sophisticated in this regard than NSW and RSEEXEC, neither of which provide such mechanisms to application programs.

7. Conclusions

As a result of this study, we have developed some conclusions about the state-of-the-art in Network Operating Systems, the capabilities required of the constituent hosts to support such distributed operating systems, and important areas for future research and development.

Current State of Network Operating Systems

We feel that the system concepts and functionality necessary for effective network operating systems are, for the most part, reasonably well understood. Research has been done in most of the relevant operating system areas so that NOSs with capabilities similar to the most advanced single site operating systems can be designed. In addition, several implemented systems such as RSEXEC and NSW have shown that many separate developments in distributed systems research can be integrated to work together in one system. At present, however, no system exists which provides all or most of the features desired for an NOS. While RSEXEC and NSW go part way in making resources distributed over a set of hosts uniformly accessible, these systems are not complete, general purpose operating systems.

The principal problems preventing development of effective network operating systems with state-of-the-art techniques lie in the areas of system performance and reliability. There are two parts to the problem: in the short term, making distributed operating systems perform and tolerate errors as well as existing single site systems are able to; and in the long term, taking full advantage of a distributed architecture to make distributed systems more reliable and perform better than centralized, single site systems.

Additionally, the use of small personal computers is becoming widespread and will become increasingly important in the future. We expect to see the basic computational resource available to a user shift from a share of a single large, general purpose, time-shared computer system to a smaller, dedicated computer system with a communications interface to a high speed local network that provides access to larger service computers as well as other personal computers. The personal computer is likely to be configured to include a variety of special purpose input/output devices such as light pens, joysticks, and bit map graphics displays. While the smaller computer systems of the future will have many of the same attributes as larger current machines, we believe there will be a need for outside support to store large volumes of data, to share data and to access unique

resources. These personal computers will be most effective when integrated into and supported by NOS systems.

Required Support From Constituent Hosts

Using existing systems as the underlying constituent hosts of an NOS merits consideration because many of the problems of providing a hospitable programming environment from a collection of hardware have already been solved.

There are, however several attributes of underlying operating systems that are required so that an expanded NOS accessing environment can be provided in a cost effective manner. First, it is desirable that the base operating system provide a convenient mechanism for augmenting the operating environment for application programs. This feature is used to create an NOS programming environment, including one in which programs that were written to run under the base operating system can also run under the expanded access provided by the NOS.

Second, the constituent host operating system should support inexpensive, long-lived processes that can run NOS component programs that provide NOS services. If an error occurs in one of these processes, there must be some way to restart the process so that the operation of the NOS can be automatic. Additional

NOS Study

Conclusion

desirable features include: the ability to specify process scheduling parameters so that important NOS services can receive expeditious service; and an efficient file system interface that operates with low overhead so that a networkwide file system built upon it can operate efficiently.

Future Research Areas

There are two general areas that appear to be important areas for research: investigation of the many practical alternatives and their implications that are encountered in the design and implementation of network operating systems; and, more theoretical study of outstanding problem areas for distributed systems.

Currently, there are many mechanisms that have been proposed as solutions to problems perceived to exist in distributed systems. In isolation, these mechanisms appear to solve the narrowly defined problems they address, but it is not clear how they would be used in a real system or how well a collection of them would work together in an integrated system. Additionally, in many cases there appear to be several alternative mechanisms that claim to solve the same problem. In short, there is still a lot of engineering required to develop complete NOS systems.

NOS Study

Conclusion

What is needed is a laboratory environment where the elements that make up an NOS can be tested to determine how well they fit together. This NOS Laboratory would permit experimentation with alternative mechanisms and prototype systems. This experimentation would permit various NOS design options to be studied without the cost commitment required to implement an operational system. NSW, with its underlying communication facility, represents a good basis for such NOS experiments. Within NSW there are several easily identifiable parts that could be used as the starting point for an NOS Laboratory. To make such a Laboratory a cost effective experimental facility, the components must be inexpensive and flexible. Anticipating developments in microelectronics, single user machines or single purpose application systems must be included as components of the experimental facility if it is to keep pace with the leading edge of technology.

One area suitable for experimental investigation in an NOS Laboratory would be techniques to support small single user computer systems by using one or more larger service machines. Before any experimentation in this area begins several important questions should be answered:

- What types of applications are suitable for such a configuration? It is very important to define classes of applications so that the types of resources accessed and the frequency of access can be characterized.
- What is the nature of the support needed by a small machine for larger machines which run an NOS like ELAN? Is there a need to move large amounts of data from the personal computer (PC) to the service computer? How fast must such transfers occur?

After these questions have been answered, the planning for such experiments could proceed.

Relevant questions that could be answered by use of the experimental facility include:

- What alternatives are there for interconnection of PCs with service computers? Which of the alternatives is most effective for the applications under consideration?
- How can an application system composed of several PCs and service computers be built so that it can grow incrementally in response to load?
- Which of several proposed file movement strategies is most effective for the applications under consideration?
- For applications that are split across a PC and a service computer, where are the best boundaries for the division?
- What kinds of reliability measures are required and which mechanisms are feasible to implement so that the target applications run in an acceptably robust manner?
- How frequently does a PC disconnect from outside communications and what are the best methods for dealing with messages directed to the PC during that time?
- Which of the several alternative levels of PC - NOS file system integration (as presented in Section 4.5) is needed for the intended application?

- What should be the division between the functionality supported by the PC and that provided by the service computer? For example, do applications require long term storage on the PC, or could this be effectively provided by a service computer connected to the PC by a very fast communication link? Should the PC be concerned only with those tasks for which it is uniquely capable, such as display management, or should it attempt other tasks such as general purpose program execution?

While many of these questions have been asked before, either in other contexts or in isolation, little is known about the integration of their answers into practical systems, and we feel that an NOS Laboratory is the most effective way to derive answers to these issues.

The second area for research should be directed toward more theoretical issues. There are several areas related to NOS design and implementation for which current knowledge is non-existent or where existing schemes are inadequate for the requirements of real applications.

Successful implementation of most distributed systems hinges on solutions to the problem of data management. Controlling multiple concurrent processes in accessing distributed data is one such problem. A number of approaches to concurrency control for a distributed environment have recently been developed. These approaches should be analyzed with respect to situations

for which they are best suited. Some of the algorithms which allow distributed processes to update distributed data were developed assuming that the requirements of applications were for random access to files. These updating algorithms turn out to be quite complex, especially when reliability is a factor.

Researchers in the field are currently seeking ways to reduce this complexity. One alternative is to recognize that the data bases used in many applications have a well defined structure, and that for many applications data access modes less general than random access read and write are sufficient. To this end, data access patterns of real applications should be closely examined in order to develop measures and models for the patterns so that application requirements can be compared with the characteristics of the various updating strategies. Other factors not captured by access pattern models, such as reliability and performance requirements, should be considered in matching the needs of distributed systems and application programs with the capabilities of existing concurrency control schemes. Finally, where mismatches between application requirements and the capabilities of existing mechanisms are identified, new mechanisms or modifications to existing ones should be developed.

In early attempts to provide access to distributed resources in the ARPANET, emphasis was placed on the most expeditious implementation. Thus, remote file access was accomplished by transfer of entire files to the referencing site where existing file access mechanisms could then directly control data manipulation. Access to remote data, even of this primitive nature, began to cause many of the traditional topics of data management to be reconsidered to take into account the characteristics of distributed architectures. Two interesting problems which arise in a distributed environment and which require further study are record level data access and data translation.

Currently, the most common way to access remote data in a network environment is to access complete files. The efficiency of such data access is low if only a small portion of the data in a file is actually accessed. Access to general purpose structured data files should be studied to determine the best ways to provide remote access to portions of files as well as entire files. Models for data structures that can be used in a network environment need to be developed. For example, how should a data base that is distributed among several sites be organized? Should intersite references from a record stored at one site to records stored at other sites be permitted? Are such

intersite references a good idea, or are there better ways to take advantage of distributed architectures? Is the conventional model where a process retrieves remote data before processing it a good one for a distributed data environment, or is an alternative model where data records remain stationary and the processing of records moves to the data site a more appropriate one?

A second aspect of remote data access in a heterogeneous environment is data translation. The problem here is to maintain the semantic content of data as it is moved from host to host. There are several approaches to data translation. The most promising appears to be standardized data types and record structure. While this represents an approach to the data translation problem, the question of what to standardize remains largely unanswered. Pursuing this approach would require the development of standards for atomic data types. An atomic data item that is to be transferred from a sender to a receiver would have to be translated from its internal format in the sender's host to a network standard format and from the network standard format to the receiver's internal format. Standards for several well established data formats, in addition to a standard standard network character set, would have to be developed. In addition, provision must be made to allow for unanticipated additions, so

NOS Study

Conclusion

that network standard data types are extendible. The idea of composite data items which are groups of atomic items should be investigated as a way for reducing the amount of application program processing that must occur when structured data is exchanged. Instead of exchanging separate atomic data items with no relation to other items, groups of items could be exchanged as a multipart record of data. Finally, as stated above, work needs to be done to standardize the structure of composite data items, or records, within a data base.

In almost all applications that have been built on a distributed architecture, little aid has been available to programmers in the form of programming language and debugging support. Research is needed to develop programming languages for use in a distributed environment. It would be helpful to relieve programmers from having to make all decisions about resource and application program component location. Formal language abstractions matched to the communication operations commonly used in distributed programs need to be developed. Linguistic concepts that support the logical unity of program modules that must execute on two or more distributed machines are required. For example, a program that manages a display screen may have to be split into two parts: a part that runs on a powerful host that can do the computation necessary to manipulate the data to

be displayed and a part that runs on a smaller host that actually drives the display screen. While there are two physical places where this program resides as it executes, a programmer would like to think of it as a single program that just happens to execute at two different sites. There should be language support for communication between the parts, perhaps in the form of shared variables or subroutine calls, so that from the programmer's viewpoint, the unity of the program is preserved.

Debugging is at present extremely difficult in a distributed environment. While some work has been done in this area [77], much more needs to be done to develop the sophisticated and flexible runtime support systems required to facilitate the debugging of distributed programs.

One problem in implementing programs, such as the components of an NOS, that must run cooperatively distributed among a collection of hosts is that of coordinating the scheduling and resource management activity of the constituent hosts with respect to the programs. For example, the current situation in the ARPANET is that processes on separate hosts, and often within the same host, are scheduled independently of each other. Although processes cooperating to perform an NOS function are not independent, each is regarded by the underlying hosts as an independent task.

It seems clear that NOS performance could be improved if some sort of interhost or global control of individual host scheduling and resource allocation actions could be exerted by the NOS. NSW is a good example of a system whose performance could be improved by global scheduling and resource allocation mechanisms. Presently the processes that comprise the NSW implementation, including system component processes for the Works Manager, Foreman and File Package and processes for tools, are scheduled independently of one another. As an illustration, when a Works Manager and a Foreman process on another host cooperate to perform an NSW operation, such as the initialization of a tool session, the two processes and those that implement the MSG communication facility for the two hosts are not related in any host resource allocation and scheduling decisions. Furthermore, there is no way for NSW to request priority for certain transactions that support interactive operations over other transactions that could occur in the background.

Global scheduling and resource allocation for distributed systems should be studied with the aim of determining useful means for expressing relationships among the components of a distributed computation and means for communicating the related scheduling and resource requirements for the components to the underlying hosts. The NOS Laboratory suggested above would be a

useful environment for evaluating various global scheduling mechanisms for distributed systems.

The potential for developing robust applications based on distributed architectures has not been widely exploited. Research is needed to develop ways to utilize the autonomy of constituent hosts and the potential for redundancy of distributed resources in order to provide reliable operation of an NOS and of the application programs it supports. The error detection and recovery mechanisms provided in ELAN are only partially developed ideas that should be extended and refined so that a program can perform application specific error detection and recovery.

One area that merits more attention is the notion of transactions. Transactions have been utilized in some data management systems as segments of computations that when taken together represent an entire task. From a reliability point of view, the attraction of performing tasks in this segmented way is that when one of the transactions of a task fails the system can be restored to its state before the transaction was attempted, and the transaction can be attempted again when there is reason to believe it will succeed. Tasks structured in this way use transactions as a form of checkpoint mechanism. The principal problem with this transaction model of computation is that not

all application programs can be transformed in a simple way into a series of transactions. Work needs to be done to incorporate the notion of a transaction into a more general computational model. In addition, techniques for transforming applications into candidate transactions need to be developed.

Finally, we believe that the meta-system approach to NOS implementation will be the predominant approach for some time to come. New single host operating systems will continue to be developed by the manufacturers, and there will be attempts to integrate these new systems into NOSs. The integration of these new systems into an NOS would be greatly facilitated and the performance of the resulting NOS greatly improved if the manufacturers could be provided with guidelines to be used at system design time for capabilities that are important or required for systems to be used in a network environment. We have identified in this report several attributes that have been useful in the implementation of several NOSs. More study is required to identify a more complete set of the important features and capabilities for single host operating systems to be used as building blocks for a meta-system NOS. Interesting questions to attempt to answer are: Given the freedom to design a "core" or "kernel" operating system that would serve as the basis for an NOS or some other form of distributed system, what

NOS Study

Conclusion

basic operations and resources should it provide? What set of data and processing resources should a manufacturer of computer systems include in a base operating system to facilitate the implementation of network operating systems?

References

TELNET/FTP

- [1] Davidson, J., N. Mimno, R. Thomas, D. Walden, W. Hathaway, J. Postel, "The ARPANET TELNET Protocol: Its Purpose, Principles, Implementation and Impact on Host Operating System Design, Proceedings of Fifth Data Communications Symposium, Snowbird, Utah, September 1977.
- [2] Feinler, E. and J. Postel (eds.), TELNET Protocol, ARPANET Protocol Handbook, 1976 Edition, pp 51-174, available from the National Technical Information Services, Accession Number ADA027964.
- [3] Ibid, "File Transfer Protocol for the ARPANET", pp 117-235.

Network Access Machine

- [4] Rosenthal, R. "Accessing Online Network Resource with a Network Access Machine," IEEE Intercon '75, section 25/3.
- [5] Manning, E.G., et. al., "A UNIX-based local processor and network access machine," Computer Networks, vol. 1, pp 139-142.
- [6] Anderson, R.H., Gillogly, J.J., "The RAND Intelligent Terminal Agent (RITA) as a Network Access Aid," NCC 1976, pp 501-508.

Distributed Computer System

- [7] Farber, David J., et. al. "The Distributed Computing System," IEEE Computer Society International Conference, COMPCON-73, 1973, pp 31-34.
- [8] Farber, D.J., Larson, K.C., "The Structure of a Distributed Computing System-Software," Computer-Communications Network and Teletraffic, April 1972, Brooklyn Poly., pp 539-545.
- [9] Farber, D.J., Heinrich, F.R., "The Structure of a Distributed Computer System - The Distributed File System," Symp. on Computer Communications, 1972, 1st International Conference on Computer Communications, pp 364-370.

- [10] Farber, D.J., Larson, K.C., "The System Architecture of the Distributed Computer System - The Communications System," Computer-Communications Network and Teletraffic, Brooklyn Polytechnic Institute, 1972, pp 21-27.

Distributed Processing System

- [11] White, J.E., "A high-level framework for network-based resource sharing," NCC, 1976, pp 561-569.

Distributed Computer Network

- [12] Mills, D.L., "An Overview of the Distributed Computer Network, NCC 76, pp 523-531.
- [13] Mills, D.L., "The Basic Operating System for the Distributed Computer Network," Computer Science Technical Report TR-416, University of Maryland, Oct. 1975.
- [14] Mills, D.L., "Dynamic File Access in a Distributed Computer Network," Computer Science Technical Report TR-415, University of Maryland, Oct. 1975.

Resource Sharing Executive

- [15] Thomas, R.H., "A Resource Sharing Executive for the ARPANET." AFIPS Conference Proceedings, Vol. 42, June 1973, pp 155-163.
- [16] Thomas, R.H., "JSYS Traps - A TENEX Mechanism for Encapsulation of User Processes," AFIPS Conference Proceedings, Vol. 44, May 1975, pp 351-360.
- [17] Cosell, B.P., P.R. Johnson, J.H. Malman, R.E. Schantz, J. Sussman, R.H. Thomas, and D.C. Walden, "An Operational System for Computer Resource Sharing," Proceedings of the Fifth ACM Symposium on Operating System Principles, published as ACM Operating Systems Review, Vol. 9, No. 5, November 1975, pp 75-81.

National Software Works

- [18] Crocker, S.D., "The National Software Works: A New Method for Providing Software Development Tools Using the ARPANET", presented at Consiglio Nazionale delle Ricerche Istituto Di Elaborazione Della Informazione Meeting on 20 Years of Computer Science, Pisa, Italy, June 1975.

- [19] Schantz, R.E. and R.E. Millstein, "The Foreman: Providing the Program Execution Environment for the National Software Works System," BBN Report No. 3442, Massachusetts Computer Associates Document No. CADD-7701-011, January 1977.
- [20] Cashman, P.M., R.A. Faneuf, C.A. Muntz, "File Package: The File Handling Facility for the National Software Works," CADD-6712-2711, December 1976.
- [21] NSW Protocol Committee, "MSG: The Interprocess Communication Facility for the National Software Works," BBN Report No. 3483, Massachusetts Computer Associates Document No. CADD-7612-2411, December 1976.

Designs for Network Operating Systems

- [22] Pierce, R.A., Moore, D.H., "Network Operating System Functions and Microprocessor Front Ends," Compcon 77 (spring 1977), pp 325-328.
- [23] Kimbleton, S.R., Mandel, R.L., "A Perspective on Network Operating Systems," NCC 1976, pp 551-559.
- [24] Kimbleton, S.R., Mandel, R.L., "Distributed Computation Study," NTIS AD-A024670, April 1976.
- [25] Manning, E.G., Peebles, R.W., "A Homogeneous Network for Data Sharing-Communications," Computer Network, Vol. 1, pp 211-224 (1977).
- [26] Peebles, R.W., Manning, E.G., "A Computer Architecture for Large (Distributed) Data Bases," Proc. First VLDB Conference, Sept. 1975, pp 405-427.

Interprocess Communication in a Network Environment

- [27] Walden, D.C., "A System for Interprocess Communication in a Resource Sharing Computer Network," CACM, Vol. 15, No. 4, pp 221-230, April 1972.
- [28] Carr, C.S., S.D. Crocker and V.G. Cerf, "Host-Host Communication Protocol in the ARPA Network," AFIPS Conf. Proc., Vol. 36, 1970 SJCC.
- [29] Cerf, V. and R. Kahn, "A Protocol for Packet Network Interconnection," IEEE Transactions on Communication, May 1974.

- [30] Akkoyunlu, E., Bernstein, A., Schantz, R., "Interprocess Communication Facilities for Network Operating Systems," Computer, June 1974, pp 46-55.

See also reference [21] on the MSG IPC mechanism.

Data Reconfiguration

- [31] Anderson, R.H., et. al., "The Data Reconfiguration Service--An Experiment in Adaptable Process/Process Communication," The ACM/IEEE Second Symposium on Problems in the Optimization of Data Communications Systems, Oct. 1971, pp 1-9.
- [32] Levine, P.H., "Facilitating Interprocess Communications in a Heterogeneous Network Environment," M.I.T. Lab. for Computer Science Technical Report MIT/LCS/TR-184.

Distributed Data Bases - General

- [33] Deppe, M.E., Fry, J.P., "Distributed Data Bases: A summary of research," Computer Networks, Vol. 1 (1976), pp 130-138.
- [34] Peebles, R.W., Manning, E.G., "System Architecture for Distributed Data Management," Computer, Vol. 11, No. 1, January 1978, pp 40-47.

Resource Allocation

- [35] Chu, W.W., "Optimal File Allocation in a Multiple Computer System," IEEE Transactions on Computers C-18:10, Oct. 1969, pp 885-889.
- [36] Mahmond, S., Riordon, J.S., "Optimal Allocation of Resources in Distributed Information Networks," ACM/TODS, Vol. 1, No. 1, March 1976, pp 66-78.
- [37] Chandy, K.M., Hewes, J.E., "File Allocation in Distributed Systems," Proc. of the Conference on Modelling and Measurement, Harvard University, Cambridge, Massachusetts, 1976, pp 10-13.
- [38] Levin, K.D., Morgan, H.L., "A dynamic Model for Distributed Data Bases," Proc. of the ORSA/TIMS Conf., Spring 1975.

Data Base Synchronization

- [39] Thomas, R. H., "A Solution to the Concurrency Control Problem for Multiple Copy Data Bases", "Proc. 1978 IEEE Spring COMPCON, San Francisco, Feb 1978".
- [40] Johnson, P.R., Thomas, R.H., "The Maintenance of Duplicate Data Base, ARPA Network Working Group Request for Comments (RFC) #677, Network Information Center (NIC), Document No. 31507, January 1975.
- [41] Stearns, R.E., Lewis, P.M., Rosenkrantz, D.J., "Concurrency Control for Data Base Systems," General Electric Research and Development Center, Schenectady, N.Y., May 1976.
- [42] Mullery. A.P. "The Distributed Control of Multiple Copies of Data," IBM Research Report RC 5782, December 29, 1975.
- [43] Alsberg, P.A., G.G. Belford, J.D. Day and E. Grapa, "Multi-Copy Resiliency Techniques," Center for Advanced Computation, Univ. of Illinois, CAC Document No. 202, May 1976.

Languages for Expressing Distributed Computations

- [44] Feldman, J.A., "A Programming Methodology for Distributed Computing (among other things)," Dept. of Computer Science, Univ. of Rochester, TR 9, no date.
- [45] Dennis, J., D. Misunas and C. Lung, "A Highly Parallel Processor Using a Data Flow Machine Language," Computer Structures Group Memo 134, MIT Laboratory for Computer Science, January 1977.
- [46] Yonezawa, A. and C. Hewitt, "Modelling Distributed Systems," 5th International Conf. on Artificial Intelligence, August 1977, pp 370-377.

Fault Tolerance

- [47] Lampson, B.W., Sturgis, H.E., "Crash Recovery in a Distributed Data Storage System," Working Paper, XEROX Palo Alto Research Center, 1976.
- [48] Ornstein, S.M., W.R. Crowther, M.F. Kraley, R.D. Bressler, A. Michel and F.E. Heart, "Pluribus- A Reliable Multiprocessor," Proc. National Computer Conference, May 1975.

Access Control and Protection

- [49] Fabry, R., "Capability Based Addressing," CACM, Vol. 17, No. 7, July 1974, pp 403-412.
- [50] Donnelley, J.E., "A Distributed Capability Computing System," Proc. Third International Conf. on Computer Communication, August 1976, pp 432-440.
- [51] Kent, S.T. "Encryption Based Protection Protocols for Interactive User-Computer Communication," MIT Laboratory for Computer Science Technical Report 162, May 1976.
- [52] Bolt Beranek and Newman Inc., "Interfacing a Private Line Interface (PLI) to an IMP and a Host to a PLI," Appendix H of BBN Report No. 1822, "Specifications for the Interconnection of a Host and an IMP," January 1976.
- [53] Diffie, W. and M.E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, November 1976.
- [54] Rivest, R., A. Shamir and L. Adelman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, CACM, Vol. 21, No. 2, February 1978, pp 120-126.

Time Sharing Operating Systems

- [55] Ritchie, D.M. and K. Thompson, "The UNIX Time-Sharing System," CACM Vol. 17, No. 7, July 1974, pp 365-375.
- [56] Bobrow, D.G., J.D. Burchfiel, D.L. Murphy and R.S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10," CACM Vol. 15, No. 3, March 1972, pp 135-143.
- [57] Corbato, F.J., J.H. Saltzer and C.T. Clingen, "Multics- The First Seven Years," AFIPS Conf. Proc. Vol. 40, 1972 SJCC, pp 571-583.
- [58] Daley, R.C. and P.G. Neumann, "A General Purpose File System for Secondary Storage," AFIPS Conf. Proc., Vol. 27, 1965, pp 213-229.
- [59] PRIMOS: The Operating System for the PRIME Computers, documentation available from PRIME Computer Corp., Framingham, Massachusetts.

Local Area Networks

- [60] Metcalfe, R. and D. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," CACM Vol. 19, No. 7, July 1976, pp 395-404.
- [61] Cotton, I. (editor), "Proceedings of a Workshop on Local Area Networks," National Bureau of Standards, (in press) August 1977.
- [62] Farber, D.J., "A Ring Network," Datamation Vol. 21, No. 2, February 1975, pp 44-46.
- [63] Mockapetris, P.V., M.R. Lyle, D.J. Farber, "On the Design of Local Network Interfaces," Proc. IFIPS 77, August 1977, pp 427-430.
- [64] Clark, D.D., "A Contention Ring Network," MIT Laboratory for Computer Science, Local Network Note 11, September 1977.

Single User Machines

- [65] Horn, B.K.P. and P.H. Winston, "Personal Computers," Datamation, May 1975, pp 111-115.
- [66] Lampson, B.W., "An Operating System for a Single-User Machine," Lecture Notes in Computer Science: Operating Systems, G. Goos and J. Hartmanis editors, Springer-Verlag, Berlin, 1974, pp 208-217.

Communication Networks and Support Services

- [67] Roberts, L.G. and B.D. Wessler, "Computer Network Development to Achieve Resource Sharing," AFIPS Conf. Proc., Vol. 36, 1970 SJCC, pp 543-549.
- [68] Hovey, R.B., "The User's Role in Connecting to Value Added Networks," Data Communications, May/June 1974.
- [69] Clipsham, W.W., F.E. Glave and M.L. Narraway, "Datapac Network Overview," Proc. Third International Conf. on Computer Communications, Toronto, Canada, August 1976, pp 131-136.
- [70] Kahn, R.E., "The Organization of Computer Resources into a Packet Radio Network," National Computer Conference, 1975, pp 177-186.

NOS Study

References

- [71] Willard, D.G., "A Time Division Multiple Access System for Digital Communication," Computer Design, Vol. 13, No. 6, June 1974, pp 79-83.
- [72] Ornstein, S.M., F. Heart, W.R. Crowther, S.B. Russell, H.K. Rising and A. Michel, "The Terminal IMP for the ARPA Network," AFIPS Conf. Proc., Vol. 40, June 1972, pp 243-254.
- [73] McKenzie, A.A, B.P. Cosell, J.M. McQuillan and M.J. Thrope, "The Network Control Center for the ARPA Network," Proc. of the First International Conf. on Computer Communications, Washington, D.C., October 1972, pp 185-191.

Task Partitioning and Synchronization

- [74] Jenny, C.J., "Process Partitioning in Distributed Systems," IBM Report RZ 873, October 1977, 35 pages.
- [75] Reed, D.P. and R.K. Kanodia, "Synchronization with Eventcounts and Sequencers," to appear CACM, Spring 1978.

Special Topics

- [76] Sher, M., "A Case Study in Networking," Datamation, Vol. 20, No. 3, March 1974.
- [77] Mader, E.R. "Network Debugging Protocol," ARPA Network Working Group Request for Comments RFC 643, July 1974.
- [78] Reed, D., private communication.

APPENDIX

State of the Art in Local Area Networks

Three important areas of concern for local networks are the architecture of the local network, the hardware technology necessary to support a local network and new issues and problems that arise in the protocols for using the local network.

At a recent Workshop on Local Area Networks [61] sponsored by the National Bureau of Standards, three popular architectures for local networks were discussed: contention bus, ring, and contention ring. For each of the architectures there are hosts (ranging from unsophisticated terminals to large computer systems), local network interfaces (LNIs) and a transmission medium (twisted pair, coax cable, free space, fiber optic cables). LNIs typically contain: a tap to the transmission medium, such as a solder joint for twisted pair, a "stinger" tap for a coax cable, an antenna for radio frequency transmission and mechanical connectors for fiber optic cables; a receiver/transmitter to encode/decode bit streams the transmission medium; and a host interface which may perform buffering for incoming messages as well as have direct access to the primary memory of the host. In addition to transmitting data, it is the responsibility of the LNI to examine every message to determine whether it is addressed to the connected host. If so, the LNI must move the message into the host. The different architectures require different actions by the LNI

regarding putting received messages back onto the transmission medium.

In the contention bus, messages are put onto the bus by the sender, and are recognized and recorded by the receiver. Because messages "die out" on the bus there is no need to explicitly remove them from it. Since two senders may attempt to put a message on the bus at the same time (the "contention"), a sender must also listen to the bus to determine whether its message was garbled by another message transmitted on top of it. Generally for contention bus architectures, messages must be explicitly acknowledged by the receiver as part of a higher level protocol.

With ring networks [62], a control token circulates around the ring. The presence of the token is an indication that the ring is idle. To send a message a node must wait until the token reaches it. The sender then takes the token off the ring and sends the message followed by a token. The message circulates around the ring until it returns to the sending node. The sender then takes it off the ring, allowing the token which follows to continue around the net. In a ring net, the LNI must "return" a received message back onto the ring so that it can return to the sender. The "round trip" taken by all messages facilitates acknowledgement of messages (they can be marked as "received" by the receiver) as well as messages that are directed to more than one host. Message acknowledgment in a ring network can be at the same low level of protocol as message transmission.

An interesting compromise between contention bus architectures and ring architectures is the contention ring [64]. One of the advantages of a contention architecture is that a host can start transmitting as soon as it wants. Of course, there is a chance that the transmission will have to be repeated due to collisions with other messages on the bus. In a contention ring, the LNI starts to send a message onto the ring whenever it wants and listens on the ring for the message. If the message received is not the message sent, then a message collision has occurred and the message must be retransmitted.

There are several hardware technologies necessary to support a local network: cable technology, transceiver design and interface design. Five different transmission mediums have been used or proposed for local networks currently being developed: twisted pair, coaxial cable, fiber optic cable, power transmission systems and free space. One general conclusion is that the cost of the physical transmission medium is insignificant compared to the rest of the equipment involved. For example, less than \$1000 was spent on all of the cable in the Ethernet of Xerox PARC, compared with \$3M on the hardware (including hosts). Twisted pair (and possibly fiber optic cables) are typically used in point-to-point architectures, such as rings. For reliability in contention bus architectures, it is desirable for the tap to be passive. Thus there is need for a low-loss tap into the medium. Coax cable was chosen for the Ethernet because of the existence of the Jerrold tap, a low cost

device made for CATV applications. The problem with fiber optic cables for contention buses is the lack of a suitable (passive) tap. When power transmission systems are used as the transmission medium, modems are used to add a signal to the normal 60 cycle signal on the power distribution system. Radio frequency transmissions are used as the medium in the Packet Radio Network [70].

A main issue in the design of a transceiver is how to transmit bit streams as signals over the transmission medium. There appear to be as many schemes as there are local networks. Ethernet uses base-band signalling on a coax cable for which a constant voltage is applied to the cable and signals for one's and zero's are added to that constant voltage. A problem with base-band signalling is that different ground potentials existing in different parts of a building tend to add noise to the transmission medium. In the Mitre-Net [71], a radio frequency transmission scheme is used to transmit signals on coaxial cable. One's and zero's are encoded in much the same way as FM radio signals. The receiver must be tuned to the proper frequency to receive the signal. Consequently, several separate contention nets can be supported on the same cable.

LNIs for existing local area networks vary greatly in cost and functionality. Cost estimates range from \$50 to \$25,000. Speeds run from 50Kbit/sec to 50Mbit/sec. Functionality ranges from the bit-serial interfaces of the Ethernet, where messages

are passed to the host as they arrive, one bit at a time, to the UCI-MIT LNI which buffers complete messages for the host. To put the cost of an LNI in perspective, consider the following facts: A standard acoustic coupler costs about \$300 and is used on terminals that cost \$1300; The LNI on the Ethernet costs about \$500 for a \$10,000 host computer; A DEC LSI-11 costs about \$600. Using the Ethernet economics as an example, the LNI for connecting an LSI-11 to a local network should be in the \$30-\$50 range.

There are several basic differences between local area networks and other types of networks. Although it is possible to take issue with any of these points for any particular instance of a local network, the following are some characteristics of many local area networks.

- High bandwidth.

The speed of the transmission medium is similar (same order of magnitude) to the speed of paths within the hosts connected to the network. The network ceases to be the communications bottleneck.

- Low delay.

The time from which a host desires to send a packet to the time when the packet is put onto the network and received by the destination host is similar to the bandwidth of the network. This implies that there is little buffering in the network. Typical local networks hold only 10's of bits at a time. For example, in a ring network, the first bit of a packet will typically come back to the source host before the last bit is sent out.

- Rich connection between hosts.

In a local net, one node typically directly addresses another host. There is no need or possibility for making message routine decisions. It is just as easy and efficient to interact with one host as any other host.

- Usually inexpensive to interface.

The transmission protocols are simple enough and the distances involved are short enough so that inexpensive LNIs can be built.

- Because of the low cost of local nets, they quite often are optimized for specific system requirements.

If an application has requirements that are not matched by any existing local network, then it is reasonable to get a new local net and tailor it to the needs of the application (of course at the expense of lacking interconnection with other applications).

Geographic distance is not considered a factor in characterizing a local network. For example, it is possible to have an ARPANET-like system contained within the boundaries of a single room. The strategies employed in this type of communications network are far from the strategies used in most local networks. The geographic closeness of hosts does not necessarily determine the locality of the net.

There are at least two impacts that high speed local networks will have on application programs. Because of the low delay of messages in the network, the protocols employed on a local network (both low and high level) will probably be simpler than those on more global networks such as the ARPANET. In particular, there will be less pipelining in protocols. In both ring nets and contention bus nets there is at most one packet on

the net at any one time -- typically a net will instantaneously hold only a fraction of a packet. Few protocols will have to be concerned with problems associated with multiple outstanding messages. For example, protocols on the ARPANET must compensate for packets that arrive out of order from when they were transmitted. In local networks with much simpler transmission schemes, messages cannot arrive unsynchronized. A second change will be an attempt to push more functionality out to other processors on the net, in effect trading the bandwidth of the net for local processing capacity. On a local network remote (process to process) interactions may be just as efficient as local interactions.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C³) activities, and in the C³ areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

